GRAPHICAL PAGE DEVELOPMENT OF AN

ELECTRONIC HORIZONTAL SITUATION INDICATOR

by

CHARLES A. ROBERTSON

B.S., Kansas State University, 1985

A MASTER'S THESIS

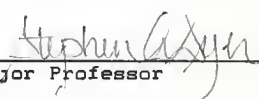submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1987

Approved by:

_____
Major Professor

TABLE OF CONTENTS

CHAPTER ONE   INTRODUCTION

CHAPTER TWO   MAIN ROUTINE

CHAPTER THREE   TOOLS FOR VECTOR GRAPHICS
                 DEVELOPMENT OF THE EHSI

CHAPTER SIX   DATA PAGE

.

TABLE OF CONTENTS CONTINUED

LIST OF FIGURES

LIST OF FIGURES CONTINUED

CHAPTER ONE

INTRODUCTION

1.1 What is an EHSI?

An electronic horizontal situation indicator (EHSI) is a digital avionics system which utilizes a cathode-ray tube (CRT) display. The EHSI provides a pilot with a pictorial representation of a plane's position relative to known navigational fixes. This representation contains distance and directional information to the fixes along with other pertinent data.

EHSI's have been developed for the larger commercial jets such as the Boeing 757 and the Airbus A310. Many of the military jets also contain EHSI's. However, no affordable EHSI system has been developed for general aviation aircraft in the lower and middle price ranges. This is surprising because most of these types of aircraft are flown by single pilots under instrument conditions. An EHSI would be ideal for such a pilot flying under instrument

1

flight rule (IFR) conditions.

The main goal of our EHSI is to provide an affordable EHSI for the general aviation public.

1.2 Purpose of the EHSI

The purpose of the Electronic Horizontal Situation Indicator (EHSI) is to ease the burden of assimilating all the data that a pilot has to make use of during the course of a flight. It will provide the pilot with three different pages of information which will be displayed on a vector graphics display and will have a control keyboard so that the pilot can enter flight parameters and control the various functions of the EHSI.

The functions which will be available on the EHSI will consist of the following:

1) FLIGHT DATA PAGE : This page will contain such information as the plane's heading, airspeed, present altitude, assigned altitude, minimum descent altitude (MDA) or decision height (DH), navigational frequencies, communication frequencies, automatic direction finder(ADF) frequency, way points, current time, temperature, etc.

2) NAVIGATIONAL PAGE : This page will provide the pilot with information about the plane's heading in a compass type format, position information with regard to known navigational fixes such as VORTAC's and non-directional beacons (NDB's), present altitude, airspeed, etc.

3) INSTRUMENT LANDING SYSTEM (ILS) PAGE : This page will provide information about a plane's position on an ILS approach. It will contain the plane's position relative to the glideslope and localizer

and display it in some useful format to allow the
pilot to do an instrument landing. Heading,
altitude, MDA/DH, and marker status will also be
displayed.

4) calculator functions : Basic calculator functions
will be available to the pilot for making various
calculations during the course of a flight.
These functions will be implemented in an HP-style
format.

5) miscellaneous functions : Other functions which will
be incorporated will be the ability to enter
various flight parameters such as the
navigational, communication, and adf frequencies,
assigned altitude, MDA/DH, and estimated wind.
The EHSI will also contain a real-time clock which
can be set with the control keyboard and a timer
which will allow the pilot to know the time that
he has been on an ILS approach.

All the pages discussed above will be dealt with in
detail in subsequent chapters. The pages are also discussed
in [1] and [2]. The majority of the information for the page
layouts was derived from these papers. Examples, of these
pages can be seen in Fig. 1.1 through 1.3.

1.3 Development System Basics

1.3.1 Goals for the EHSI Development System

The EHSI Development System has six basic goals which
it needs to accomplish. These goals are the following:

1) provide access to pertinent flight data from flight
simulator

2) retrieve control keyboard inputs, execute operation
or send input to host

```
HEADING:   217
AIRSPEED:  153 CAS
           TAS
GNDSPEED:              500
                    C
                    L
ASSIGNED:  3000     I
ALTITUDE:  3040     M
MDA/DH:    1229     B

TIMER:     05:39
TIME-OUT:  05:50

CDM1:  119.10        TIME:
CDM2:  121.90        EDT          12:05:06
NAV1:  112.6         ZULU
NAV2:  110.1         SINCE L/D

RNAV: WP1            ADF: 242
      WP2

TEMP:                BAROMETER:
```
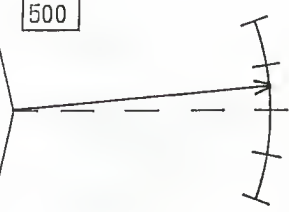
Figure 1.1 DATA PAGE

4
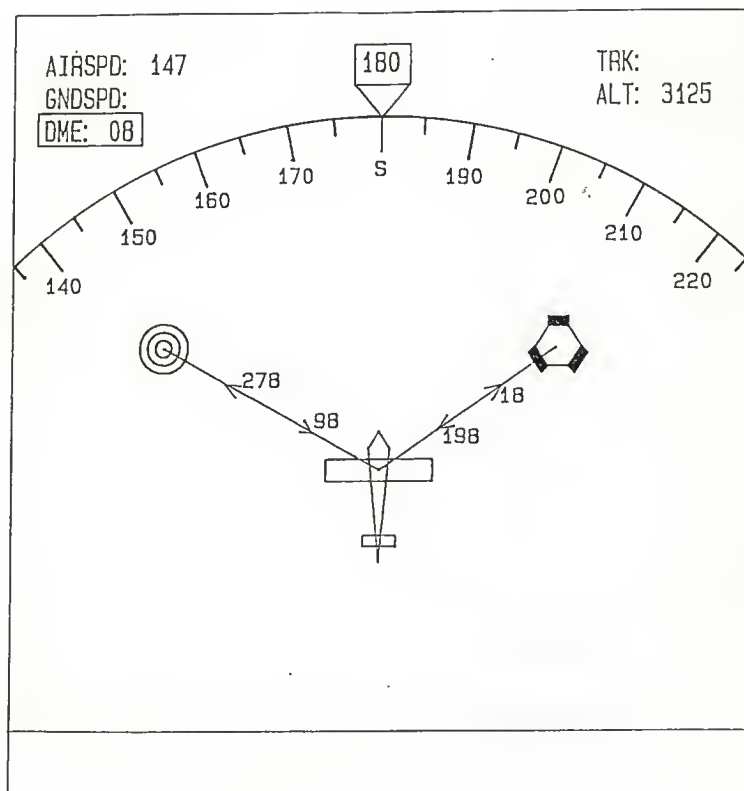
Figure 1.2 NAVIGATIONAL PAGE

5
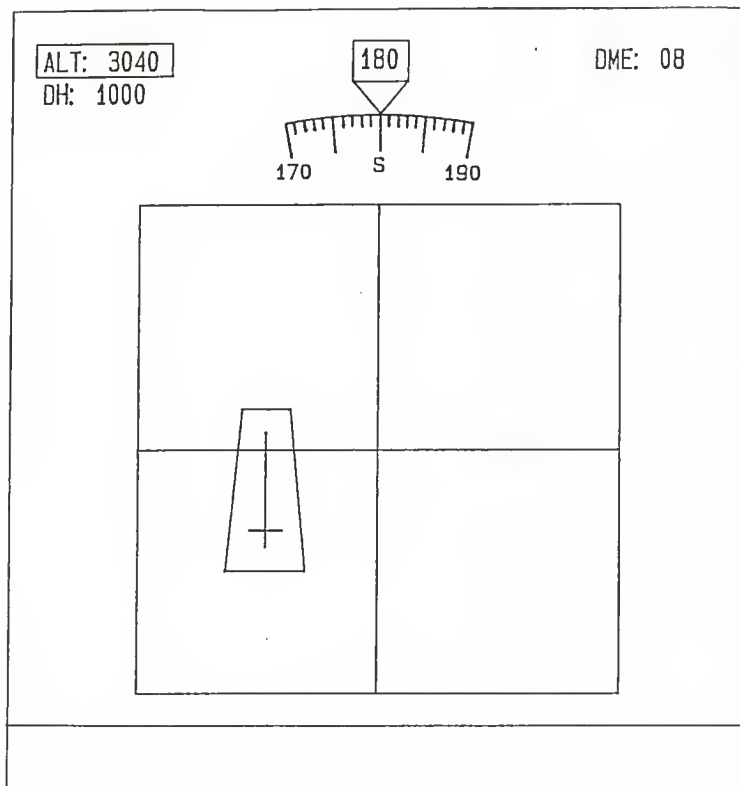
Figure 1.3 INSTRUMENT LANDING SYSTEM (ILS) PAGE

3) provide alarm capability

4) provide screen data transfer capability

5) have a simple communications protocol between interface and host

6) graphics development and data manipulation are to be done on host computer

1.3.2 EHSI System Components

The EHSI Development System consists of five basic components which can be seen in Fig. 1.4. The five components are the following:

1) Data Acquisition and Communications Interface (DACI)

2) Control keyboard with system on/off switch

3) HP 1345A Vector Graphics Display (VGD)

4) ATC-610 flight simulator

5) Host computer : Zenith 158-A PC

The DACI is a Motorola MC68000-based system which makes use of a Motorola MC68000 Educational Board as the controller. The DACI provides the means by which all the other system components talk with each other. This interface was necessary because those components are stand-alone devices and have no otherwise effective means of communicating with one another. The DACI acquires all the analog and binary signals from the flight simulator. The DACI also conditions all of these signals, converts them to digital values, and

7
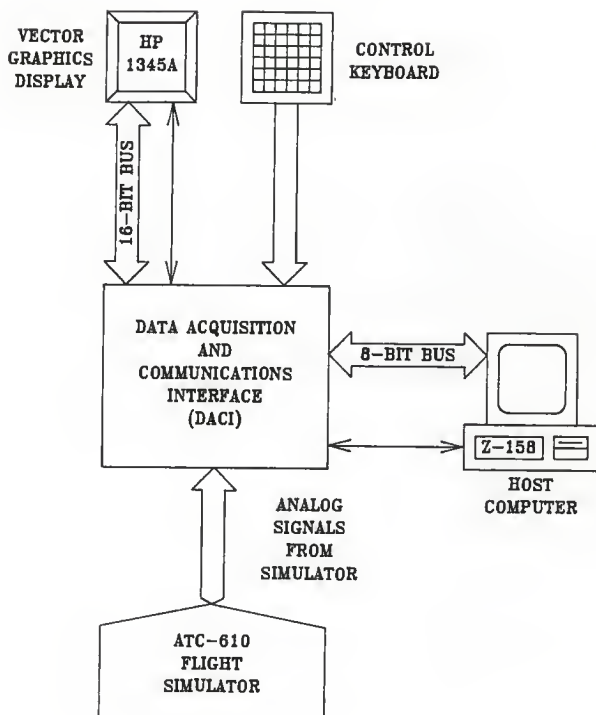
Figure 1.4   DEVELOPMENT SYSTEM COMPONENTS

8

stores them away so they are available to be sent to the host computer when requested. The host computer will then process these digital values. The DACI has a real-time clock and calendar on it. The real-time clock value is also sent with the converted flight simulator signals when re-quested by the host. The DACI processes all key presses, including the state of the system on/off switch, and sends them to the host for evaluation. The DACI sends display commands to the HP 1345A VGD and allows the host to transfer display commands to or from the VGD. The DACI contains a piezo-electric alarm which can be toggled by the host. The DACI also performs all the necessary initializations for the display. This initial system was designed by Lagerberg and more details about the DACI can be found in [2].

The commands accepted by the DACI are the following:

1) send data package

2) send screen data to the VGD

3) receive screen data from the VGD

4) toggle alarm

All command requests are sent by the host to the DACI via a parallel port. The commands allow the host to get the flight data from the simulator, process it, generate dis-plays, and send them to the VGD. All processing for the displays is done by the host; the DACI just acts as an intermediary between the host and the VGD. The alarm is

used to alert the pilot of impending conditions which may be threatening.

The control keyboard, shown in Fig. 1.5, is a six-by-six matrix keyboard with system on/off switch. This keyboard is used to determine system status (on or off) and to allow entry of various commands or parameters. These parameters include such things as navigational frequencies, communication frequencies, ADF frequency, timer, assigned altitude, MDA/DH, and estimated wind. These are entered by the pilot either during preflight or during the course of a flight. The commands which can be entered by the keyboard are "start timer," "reset timer," basic Hewlett-Packard(HP)-style calculator functions, "clear," "enter," "toggle alarm," "set clock," etc. The three different page keys are used to switch among the various pages displayed by the EHSI. It should be noted that when the system switch is turned off, the system enters a wait state. When the system switch is turned back on, the whole system is reinitialized and is set up in its original state, ready for commands from the host and for retrieval of data from the flight simulator.

The HP 1345A is a high-resolution vector graphics display. It is used to display all the graphics generated by the interface and host. It has a 2048-by-2048 addressable area for plotting. It is especially suited for this type of application since it is capable of operating up to 15,000

10

EHSI
SYSTEM
SWITCH

ON

OFF

START TIMER	BRG/HLD INBND		TIMER	RESET TIMER	SET CLOCK
ADF	FLIGHT DATA PAGE	MDA/DH	ASGN ALT	EST WIND	SET/RST ALARM
VOR1	RNAV PAGE	—	7	8	9
VOR2	ILS PAGE	+	4	5	6
COM1	CLEAR	X	1	2	3
COM2	ENTER	÷	0	.	/

Figure 1.5   CONTROL KEYBOARD

11

feet pressure altitude.   It accepts four basic commands and has a built in character set.   The display and its various commands and attributes will be discussed in more detail in Chapter 4.

The ATC-610 flight simulator is used to obtain all the pertinent flight data.   It is an FAA approved instrument flight simulator.   All of the analog signals from the simu- lator are conditioned by the DACI and converted into digital values which can by requested by the host.

The host computer that was chosen for this system was the Zenith 158-A PC. It is responsible for manipulating the data retrieved by the DACI and generating all the graphics for the different pages of the EHSI.   The C programming language was chosen; it has many high-level language fea- tures, can still do bit manipulations very easily, and is a fast-running language.   The Microsoft C Compiler, Version 4.0, was used.

The Z-158 used has a standard RS-232C serial asynchron- ous communications port, a Centronix standard parallel printer port, a CPU clock speed of either 4.77 Mhz or 8 Mhz, an 8087 numeric co-processor, and additional backplane exp- ansion slots. The computer is configured with 640k bytes of RAM.   It has one floppy disk drive and a 20-megabyte Win- chester hard disk system.   It should be noted that all the routines are run with the Z-158 set at the 8 MHz clock rate

to keep processing time to a minimum.

The Centronix standard parallel port is used as the general purpose I/O port. It should be noted that hardware modifications were made to this port and are discussed by Lagerberg in [2]. Since the standard parallel port drivers have a limited transfer rate of 1000 characters per second, parallel port drivers were written in 8088 assembly language to communicate at 10,000 characters per second with the DACI. This transfer rate was need for the development system. These routines were written by Gruenbacher and are discussed in detail in [3]. These drivers are accessed by the C routines used to generate the displays.

The Z-158 operates in an interrupt environment and makes use of the parallel port autovectored interrupt number seven. This interrupt is actually installed as a background process and is discussed in detail in [3]. The background interrupt process retrieves all interrupts sent by the DACI and stores them in a 20-byte stack. The stack is needed so that no interrupts will be missed.

1.4 Present Status of the EHSI Development System

1.4.1 Previous Accomplishments

Lagerberg designed the initial development system which satisfied the six basic goals stated previously. The initial system accomplishes the following:

13

1) provides access to pertinent flight data from flight
   simulator

2) retrieves control keyboard inputs, execute operation
   or send input to host

3) has alarm capability

4) provides screen data transfer capability

5) has a simple communications protocol between
   interface and host

6) allows graphics development and data manipulation to
   be done on host computer

This system involves a combination of hardware and software
to accomplish its duties.   Details about this portion of the
development system can be found in [2].

1.4.2 EHSI Host-side Development: Gruenbacher's Accomplishments

Gruenbacher developed five separate areas of the host-
side of the development system:

1) background interrupt process to accept interrupts
   from the DACI

2) interface communications routines in 8088 assembly
   language

3) interface of the assembly routines with the C
   programs

4) portions of the main routine of the EHSI program

5) key routines of the EHSI program

14

The background interrupt process retrieves all interrupts sent by the DACI and stores them in a 20-byte stack. The stack structure was implemented so no interrupts would be missed while display data is being generated in the EHSI routines. This interrupt process is installed when the EHSI system switch is turned "on". Note, the EHSI program must running before the system switch is turned "on".

The interface communications routines allow the host computer to talk with the DACI. They are written in 8088 assembly language and handle all the communications with the DACI. They also allow the EHSI program to use the four basic DACI commands which were discussed previously.

The 8088 assembly routines had to be interfaced with the EHSI program which is written in C. This involved using several public data structures which could be accessed by both the assembly and C routines. These public structures were necessary so that both the assembly routines and the C routines could access them. There are two main public structures which were used. The first one contains the flight simulator data collected by the DACI. It is called the data_pkg. The second contains the screen data to be sent to the VGD. It is called SCREEN and is basically a single dimensional array of display commands in word form.

All the assembly communications routines can be accessed by the C routines. These include the routines which send

commands to the DACI. The DACI command routines are which can be accessed are the following:

1) GET_DATA_PACKAGE()

2) SEND_SCREEN()

3) TOGGLE_ALARM_SWITCH()

4) RECEIVE_SCREEN()

The GET_DATA_PACKAGE routine requests the DACI to send the twenty-four flight data values retrieved by the DACI. As the data values are transferred from the DACI to the host they are stored in the data_pkg structure which was mentioned previously. Once this has been accomplished any C routine can access the flight simulator data.

The SEND_SCREEN routine requests the DACI to accept the vector graphics display data in the SCREEN structure and transfer the data to the VGD. This display data which is generated by the C routines consists of the necessary commands to generate the various graphics needed for the different pages of the EHSI. The routine sends the data a word at a time until an hexadecimal value of FF is encountered in the upper byte of a screen data word. The routine then sends the word with FF in the upper byte to the DACI to indicate to the DACI that this is the end of the screen data being sent. The SEND_SCREEN routine then terminates.

The TOGGLE_ALARM_SWITCH routine requests that the DACI toggle the bit which activates or deactivates the piezo-

16

electric alarm on the DACI. This routine is used when enabled alarm conditions are impending. This alarm will indicate to the pilot that an alarm condition exists. It will be coupled with a warning message to indicating the cause of the alarm.

The RECEIVE_SCREEN routine requests the DACI to send a portion of the display memory. The portion of the display memory to be read is sent by the routine to the DACI. It consists of a beginning address and ending address of the block of the block to be read. As the block of memory is transferred by the DACI a word at a time it is stored in the SCREEN data structure. This data can then be accessed by the C routines for purposes of analysis.

The main routine is the controlling routine for all the vector graphics development of the EHSI. It is written in C and performs several basic functions. First, the background interrupt process which was discussed previously is installed. Second, initialization of data structure values passed to other routines is accomplished. Third, the static data for the various pages of the EHSI is sent to the VGD. Fourth, the data page is put as the first page to be displayed. Fifth, the routine watches for interrupts received from the DACI by the background interrupt process. Sixth, it evaluates the interrupts received from the DACI. If they are valid interrupts, it takes appropriate action to execute

17

the DACI interrupt.   Note,   the main routine and its   func-
tions will be discussed in more detail in Chapter 2.

The   key routines implement the DACI interrupts   gener-
ated by the control keyboard.   These routines are written in
C and consist of the following:

  1) update_key_buffer : This routine converts the DACI
     interrupts for the numbers 0 - 9 and the decimal
     point into characters and stores them in a buffer.
     It also displays the buffer on the VGD.

  2) roll_stack : This routine pushes the present value
     in the key buffer onto the two number HP-style
     stack used for the calculator functions.

  3) clear_stack : This routine clears out both locations
     of the HP-style stack used for calculator
     functions.

  4) insert_new_freq : This routine enters the commun-
     ication frequency, navigational frequency, or the
     ADF frequency which is in the key buffer and
     stores it in the proper frequency variable.   If
     the frequency entered is out of range, an error
     message is displayed on the VGD.

  5) set_timer : This routine enters the timer value in
     the key buffer and stores it in the timer var-
     iable.   Note, the timer value is entered in min-
     utes and seconds with a decimal point between
     them.   The routine separates the number into min-
     utes and seconds.

  6) reset_alarm : This routine is used to toggle the
     alarm on the DACI.   It is just like an on/off
     switch for the alarm.

  7) do_math : This routine implements all the math func-
     tions that are available on the control keyboard.
     The number which is presently on the bottom of
     the stack is the first operand and the number in
     the key buffer is the second operand.   The oper-
     ation to be performed is implemented on the two
     operands and the result is displayed on the VGD.
     Note, if the denominator is equal to zero when a
     divide operation is being implemented, an appro-

priate "zero divide error" is displayed on the
VGD.

8) display_data_page, display_nav_page,
display_ils_page :
These routines change the internal jump in the
first word of the vector memory of the VGD.
The jump is changed to point to the appropriate
section in vector memory which contains the infor-
mation used to generate the graphics for the page
control key that has been pressed. Thus, this
routine switches among the various pages displayed
by the EHSI, depending on what page control key
has been pressed.

These routines along with the other areas of the host-
side developed by Gruenbacher will be discussed in more
detail in [3].

1.4.3 EHSI Host-side Development:
Robertson's Accomplishment's

Robertson's portion of the host side of the EHSI devel-
opment system consisted of developing all the necessary code
to generate the various pages of the EHSI. This involved
doing the following:

1) Data structures passed to the routines that generate
the various pages of the EHSI were developed.

2) Basic tools used to generate the vector graphics
displays for each page were developed.

3) The static and dynamic portions of the FLIGHT DATA
PAGE were developed.

4) The static and dynamic portions of the NAVIGATIONAL
PAGE were developed.

19

5) The static and dynamic portions of the ILS PAGE were developed.

6) Command line and warning areas were developed for each of the pages for displaying appropriate warning messages, entering pertinent flight data, and using the calculator functions of the control keyboard.

All of the above areas will be discussed in detail in subsequent chapters, but a brief overview of each of the areas follows.

Data structures were needed to pass information between the various routines of the EHSI program. There are two structures at the present time which are being used. The first structure is is the clock package. It contains the information which is used by the timer function of the EHSI and the frequencies, altitudes, and estimated wind values which can be entered by the pilot via the control keyboard. The second structure is the alarm package. It contains the various alarm flags which are used to indicate the status of the various warning and alarm conditions which are entered by the pilot.

To generate the various pages of the EHSI some basic tools had to be developed. These tools do some of the general things which are needed on all of the pages. For example, a routine was written to insert data which has been generated into the screen data structure which was discussed

earlier.   Another   routine generates a line given two   sep-
arate   coordinates.   There are many other tools which   were
developed and they are discussed in a Chapter 3.

The FLIGHT DATA PAGE was developed in two different
parts. First, the static portion of the page was developed.
The   static portion contains those portions of the DATA PAGE
which will not change. This includes all the various titles
for the information being displayed and certain portions of
the climb rate indicator.   The routine which generates this
data is run only once when the EHSI system is powered up.
The data generated is stored in the vector memory of the VGD
at a certain location.   Next, the dynamic portion of the
data page was developed.   This portion contains all the
various information which can change during the course of a
flight.   For example, the air speed and altitude are gen-
erated by the dynamic routine and displayed next to their
appropriate titles which were generated by the routine which
generates the static information.   It should be noted that
the dynamic routine has to manipulate the data which is
collected by the DACI and convert it into value which is
usable.

The NAVIGATIONAL PAGE and the ILS PAGE are also divided
into static and dynamic portions.   Once again the routines
used to generate the static data are only run once upon
power up of the EHSI system.   This static data is stored in

21

its proper location in the vector memory of the VGD. The dynamic portions of these two pages are much more complicated than that of the FLIGHT DATA PAGE. The algorithms and procedures for these portions will be discussed in subsequent chapters of the thesis. These pages involve converting data received from the DACI, analyzing the converted data, and generating the appropriate vector display data to produce the graphics for each of the pages.

Command line and warning areas are used to display pertinent information to the pilot. These areas are slightly different for each page but contain the same basic information. This information includes appropriate warning messages when conditions occur which warrant the pilot's attention. Also, flight parameters entered by the pilot on the control keyboard are displayed along with entries which are to be operated on by the calculator functions of the EHSI.

1.4.4 EHSI Host-side Development: Robertson and Gruenbacher

There were a few things which were added to the EHSI through a collaborative effort by Gruenbacher and myself. First, dedicated interrupt lines were added to the system. This involved both hardware and software modifications to the DACI and software modifications to the host. The changes to the DACI appear in Appendix A. Second, the clock interrupt routine of the DACI was modified to analyze what

22

would need to be done to speed up the EHSI system. This involved changing the main routine of the DACI to retrieve the clock data every time in its main loop rather than on an interrupt from the real time clock, disabling the real time clock interrupt, and removing the interrupt handshaking from the clock interrupt routine. The results of this test showed that a faster processor is needed on the host-side and that a key decoder with latched outputs is needed for the control keyboard. These changes were then removed and the DACI routines were returned to their original state. Third, a switch was added to the DACI to allow the input of an external clock to the clock interrupt line of the DACI. This was also added to analyze the speed of the EHSI system. It also served to demonstrate the operation the interrupt stack on the host side. Finally, as a temporary cure for noise problems, the flat cables of the EHSI system were twisted and wrapped in a foil shield, which was grounded to the chassis of the host computer.

1.5 EHSI Host-side Programming Model

The functional programming model for the EHSI consists of five basic levels:

1) BACKGROUND INTERRUPT PROCESS

2) INTERRUPT STACK

3) MAIN ROUTINE

4) EHSI FUNCTION MODULES

5) COMMUNICATION ROUTINES

These levels can be seen in Fig. 1.6.   The paths between these different levels can also be seen in the figure.

```
┌─────────────────────────┐
│ BACKGROUND INTERRUPT    │
│ PROCESS: Retrieves      │
│ interrupts from DACI    │
└─────────────────────────┘
            │
            ▼
┌─────────────────────────┐
│ INTERRUPT STACK:        │
│ Contains interrupts     │
│ received by BACKGROUND  │
│ INTERRUPT PROCESS       │
└─────────────────────────┘
            │
            ▼
┌──────────────────────────────────────────────┐
│ MAIN ROUTINE:                                 │
│ 1) Installs BACKGROUND INTERRUPT PROCESS      │
│ 2) Evaluates all DACI interrupts and          │
│    executes appropriate actions               │
│ 3) Removes BACKGROUND INTERRUPT PROCESS       │
└──────────────────────────────────────────────┘
       ▲          ▲          ▲          ▲
       ▼          ▼          ▼          ▼
  ┌────────┐  ┌──────┐  ┌──────┐  ┌──────┐
  │ MISC.  │  │DATA  │  │ NAV  │  │ ILS  │
  │FUNCTIONS│ │PAGE  │  │PAGE  │  │PAGE  │
  └────────┘  └──────┘  └──────┘  └──────┘
       ▲          ▲          ▲          ▲
       ▼          ▼          ▼          ▼
     ┌─────────────────────────────────┐
     │ COMMUNICATION ROUTINES:         │
     │ Written in Intel 8086           │
     │ assembly language               │
     └─────────────────────────────────┘
```

Figure 1.6 EHSI HOST PROGRAM MODEL

25

CHAPTER TWO

MAIN ROUTINE

The main routine is the controlling routine for all the vector graphics development of the EHSI. It is written in the C programming language and has to perform several basic functions. This routine continually loops once it is started and terminates when the EHSI system switch is turned "off."

2.1 Functions

The first function performed by the main routine is the installation of the background interrupt process. The background interrupt process retrieves all the interrupts sent by the DACI and stores them in a twenty-byte stack. This stack decreases the possibility of missing a DACI interrupt while display data is being generated by the functions called by the main routine.

Second, all the data structure values passed to other routines are initialized. This includes clock package val-

26

ues such as the timer minutes and seconds, time-out minutes and seconds, ADF frequency, communication frequencies, navigation frequencies, timer status, and timer operation. Alarm conditions in the alarm package are also initialized to the off state.

Note, at this point, the main routine enters a continuous loop and waits for the "system on" interrupt to be sent by the DACI. The main routine is executing the clock interrupt at this time, but the static information for the different display pages has yet to be installed. Therefore, nothing is displayed on the vector graphics display.

Third, when the "system on" interrupt is sent by the DACI, a system on message is displayed and the static data for the different pages to be displayed is installed in the vector graphics memory. The term static is used here to indicate display data that will not change for the various pages. This includes such things as headings for the various flight data that are to be displayed. The DATA PAGE of the EHSI is now displayed on the VGD. This page will always be the first page shown when the "system on" interrupt is received from the DACI.

Fourth, the main routine continues looping and checks for interrupts from the DACI. When an interrupt is received it is then evaluated and checked for validity. There are three types of valid interrupts and they consist of the

27

following:

    1) key interrupts
        A) function keys
        B) number keys
        C) page keys

    2) clock interrupts

    3) system status
        A) system on
        B) system off

Key interrupts are generated when a key is pressed on the control keyboard of the EHSI. The function key interrupts are used to execute the various functions which are available. These functions include entering the ADF frequency, communication frequencies, navigational frequencies, timer functions, calculator functions, etc. When an interrupt is evaluated to be a key interrupt, the appropriate function used to execute the key interrupt is called by the main routine. Upon completion of the key interrupt routine, control is returned to the main routine. Number key interrupts are used to enter numerical values into a buffer. These numerical values in the buffer are then used when the various function keys of the EHSI are pressed. Page key interrupts are used to determine which page is to be displayed by the EHSI. Remember, the system first displays the DATA PAGE. However, after the "system on" interrupt is received, any of the three pages available on the EHSI may be selected by pressing the appropriate page key on the control keyboard.

Clock interrupts are interrupts generated by the DACI's real-time clock. Presently, the interrupt occurs every half second. This corresponds to a frequency of two hertz. When a clock interrupt is received, the routine which generates the dynamic data for the present page being displayed is called. The term dynamic here is used to refer to the vector graphics commands which are subject to change for the various displays.

System status interrupts are generated when the system switch of the EHSI is turned "on" or "off". As was stated earlier, the "system on" interrupt results in the installation of all the static data for the various pages into the VGD memory. When the "system off" interrupt is received from the DACI, the main routine removes the background interrupt process which was installed, displays a system off message, and exits to DOS.

The basic functions of the main routine can be seen in Fig. 2.1.

2.2 Data Structures

The main routine has four different data structures which it accesses. Two of these structures are public structures and two are local to the main routine. The reason for the use of the public structures was to allow the interface of the C routines and the assembly routines of the

29
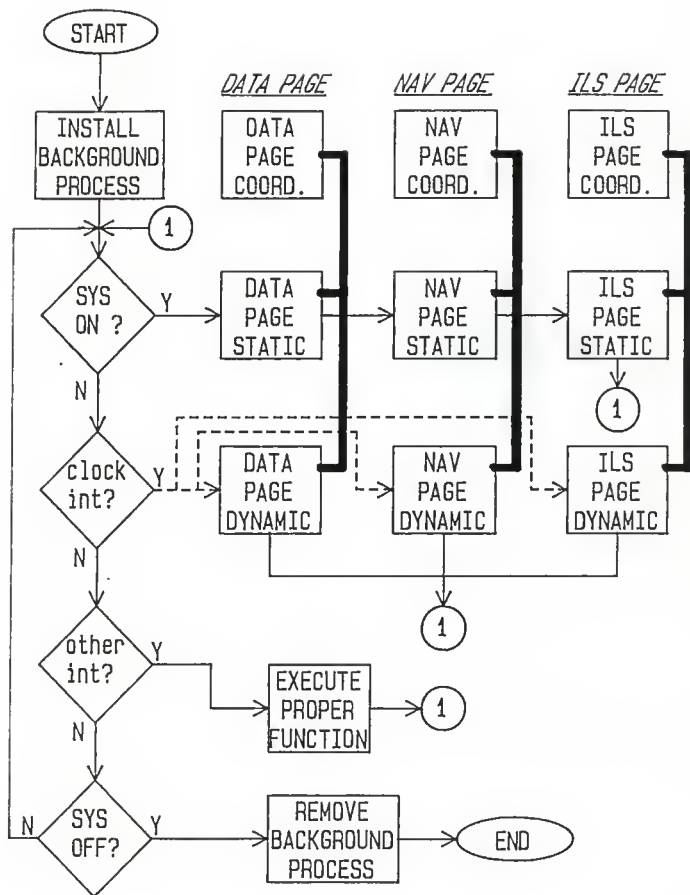
Figure 2.1   MAIN ROUTINE BASIC FUNCTIONS

30

host-side.

The first of the public data structures is the data package. This structure contains all the raw digital flight values obtained by the DACI from the flight simulator. There are a total of twenty-four different values. These values are manipulated and converted by various routines to generate the various pages of the EHSI. They are retrieved by calling the assembly routine, GET_DATA_PACKAGE(), which was mentioned in the introduction.

The second of the public data structures is the screen data. This structure contains all the VGD commands which are to be sent to the VGD. It is one-dimensional array consisting of up to one thousand different display commands. These commands are sent by calling the assembly routine, SEND_SCREEN(), which was mentioned in the introduction.

The third data structure is the clock package. This structure is local to the main routine and contains a variety of values which are passed by the main routine to various functions which it calls. This structure contains the following:

    1) time values
        A) timer minutes and seconds
        B) time-out minutes and seconds
        C) timer operation flag
        D) timer status flag

    2) frequency values
        A) ADF frequency
        B) communication frequencies
        C) navigation frequencies

3) altitude values
           A) assigned altitude
           B) MDA/DH

        4) estimated wind direction

        5) math operation flag

   The fourth structure is the alarm package. This struc-
ture is also local to the main routine and is passed by the
main routine to other routines. This structure contains the
alarm flags for the EHSI.   Presently there are alarm flags
for the airspeed, assigned altitude, MDA/DH, and the time-
out. The structure also contains alarm enable flags for the
MDA/DH and assigned altitude and a system alarm status flag.
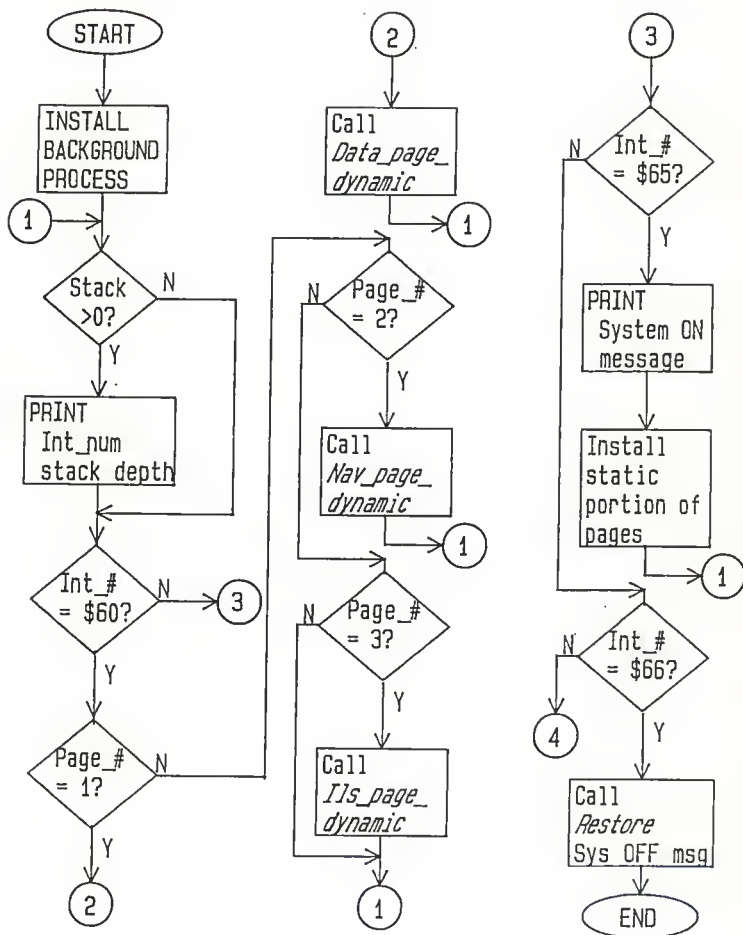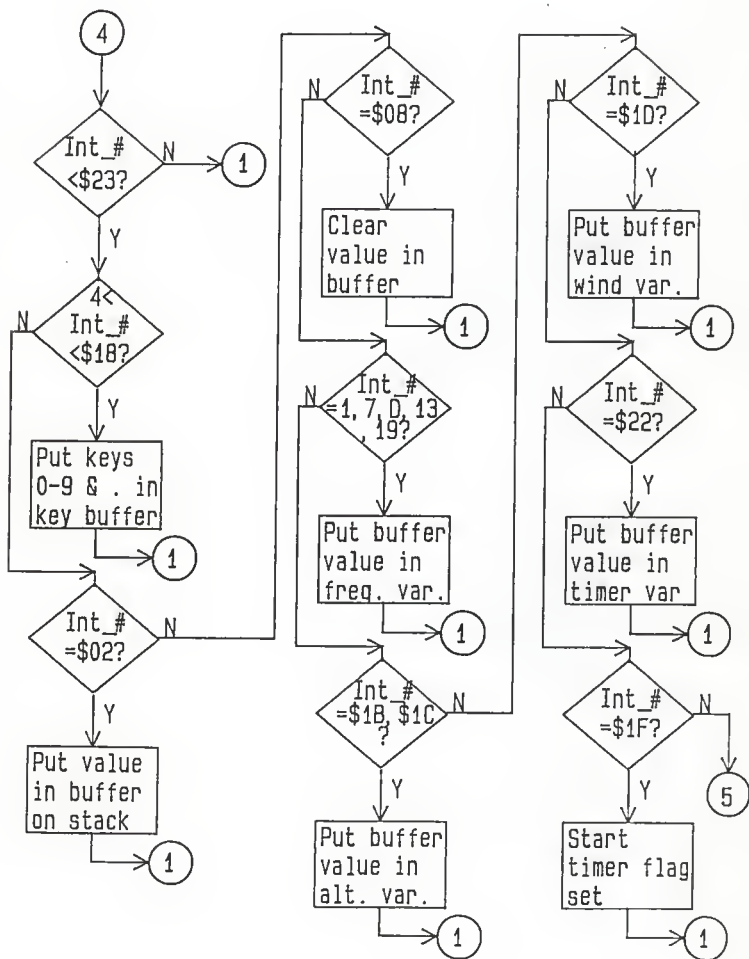
2.3 High Level Flowchart
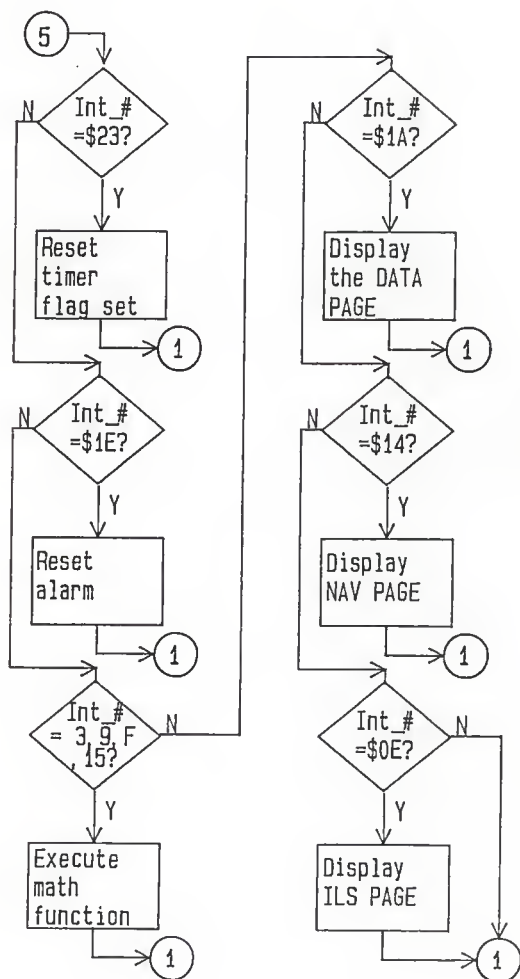
CHAPTER THREE

TOOLS FOR VECTOR GRAPHICS
DEVELOPMENT OF THE EHSI

The first step in developing the vector graphics dis-
plays for the EHSI was the development of some basic gra-
phics tools. These tools were needed to facilitate the task
at hand. These tools consist of functions which were writt-
en in the C programming language. They accomplish a variety
of separate tasks and can be divided into two different
categories. The first category consists of those functions
which have a general-purpose nature. These functions or
tools are used in the development of the graphics for all
the pages of the EHSI. The second category consists of
those tools which are page specific. These page-specific
tools accomplish tasks which are unique to the vector gra-
phics development of specific pages of the EHSI.

3.1 General-Purpose Tools

There are many different general purpose tools which

were developed to help generate the different pages of the EHSI. For the most part, these tools were the first functions that were written to ease the burden of the vector graphics development for the EHSI. Presently, these tools consist of the following:

1) **insert**

2) **line**

3) **arc_circ**

4) **string_gen**

5) **box**

6) **zero_pad**

7) **altitude**

8) **dme**

9) **airspeed**

10) **arrow**

11) **heading**

12) **timer**

The **insert** function inserts a conversion array into the screen data structure which was discussed earlier. It inserts this array based on an index which is passed to the routine. The conversion array contains the VGD commands which have been generated by other functions. The routine only inserts a certain number of commands in the conversion array. The number of commands inserted is determined by a length passed to the routine. This routine is used more

than any of the other general routines.

The **line** function generates the necessary VGD commands to plot a line between two points on the vector graphics display. The coordinates of the end-points are passed to the routine and the VGD commands which are generated are stored in a conversion array.

The **arc_circ** function generates the necessary VGD commands to generate an arc between zero and three hundred sixty degrees. The plot origin for the arc is at the center of the arc. The arc is drawn at a certain radius. The routine generates this arc by drawing many separate line segments. The angle increment for drawing the line segments is determined by the radius, with a larger radius having a smaller angle increment. This is necessary to make the arc appear to have a smooth curvature. The VGD commands generated are stored in a conversion array. The plot coordinates, start angle, end angle, and radius are all passed to the routine.

String_gen is a function which generates the necessary VGD commands to place a string at a given plot coordinate on the VGD. It can generate characters of four different sizes. The character size, plot coordinate, and string are all passed to the routine. The commands generated are stored in a conversion array. This routine is used extensively.

Box is a function which used to generate a box around a given string being plotted on the display. The size of the box generated is based on the number of characters in the string and the size of the characters. The location of the box is based on the plot location of the string on the display. The plot coordinates, character string, and character size are all passed to the routine. All the VGD commands generated are stored in a conversion array.

The zero_pad function is used to zero pad values which are to be displayed. Zero padding is just the addition of zeros to a value to maintain the number of digits shown at a constant. The zeros are added to the front of a number. For example, since a compass can have a heading of up to 360 degrees, it might be desirable to have the heading displayed with a three digit field. In the case of a heading of 185 degrees, no zero padding would be necessary. However, a heading of 5 degrees would be zero padded to 005 degrees. This function takes a given number to be padded , zero pads the number based on a given pad limit, and converts the padded value to a string. The value to be padded, the pad limit, the precision, and the type of number to be padded are all passed to the function. The padded value is stored in a conversion string.

Altitude is a function which retrieves the digital altitude value from the data package structure and converts

this value using a linear conversion into feet. This converted value is zero padded and is plotted as string at a given location on the VGD. This routine also monitors the alarm conditions for the assigned altitude and MDA/DH. It will control the alarm horn of the EHSI and will display appropriate warning messages when necessary. The clock package, alarm package, and plot coordinates are passed to the routine. The VGD commands generated are stored in a conversion array.

Dme is a function which retrieves the digital DME value from the data package structure and converts this value using a linear conversion into nautical miles. This converted value is zero padded and is plotted as string at a given location on the VGD. The clock package, alarm package, and plot coordinates are passed to the routine. The VGD commands generated are stored in a conversion array.

Airspeed is a function which retrieves the digital airspeed value from the data package structure and converts this value using a linear conversion into knots. This converted value is zero padded and is plotted as string at a given location on the vector graphics display. This routine also monitors the alarm conditions for the stall airspeed of the aircraft. Later, a structural-limit airspeed may be added. It will control the alarm horn of the EHSI and will display appropriate warning messages when necessary. The

clock package, alarm package, and plot coordinates are pass-
ed to the routine. The VGD commands generated are stored in
a conversion array.

Arrow generates the necessary VGD commands to plot an
arrow head on the display. The arrow is plotted for a given
radius, location, angle, and direction. The direction de-
termines whether or not the arrow head points "to" or "from"
the given plot location. The commands generated are stored
in a conversion array.

Heading retrieves the digital value for the plane head-
ing from the data package and converts it into an appro-
priate plane heading in degrees. It accomplishes this conv-
ersion by using a lookup table which is in the form of a
structure. If the value retrieved does not equal a value in
the lookup table, interpolation is used to get the appro-
priate plane heading. The plane heading found is passed to
the calling routine.

Timer implements the TIMER and TIME-OUT function avail-
able on the EHSI. The routine keeps track of the timer
value and compares it to the time-out value which is entered
by the operator via the control keyboard. The routine
reacts to all key presses dealing with the timer. These
include the "TIMER" key, "RESET TIMER" key, and "START
TIMER" key of the control keyboard. When the timer value is
greater than or equal to the time-out value which has been

entered, the alarm horn of the EHSI is sounded and an appro-
priate message is displayed. The only page which displays
the timer and time-out value is the DATA PAGE. The routine
stores all generated vector graphics commands in a con-
version array. The clock package and the alarm package are
required by the routine.

3.2 Page-Specific Tools

Each of the pages of the EHSI has certain portions
which are unique to that particular page. To generate these
unique graphics, functions had to be written to accomplish
these tasks. These functions are page-specific, which means
that they are only used in the development of a graphical
portion of one specific page of the EHSI. The page-specific
tools can thus be broken down into categories defined by the
EHSI pages.

3.2.1 DATA PAGE Tools

The DATA PAGE tools were the first page-specific tools
which were developed. For the most part, they are very
simple routines because the graphics generated for the DATA
PAGE are not that complicated. The page-specific tools for
the DATA PAGE consist of the following:

1) time_string_gen

2) climb_box

3) **clim_hsh**

4) **clim_arrow**

5) **climb_rate**

The **Time_string_gen** routine is used to generate a string depicting the time in hours, minutes, and seconds. The routine retrieves the values of the real-time clock on the DACI from the data package. These values, which consist of the hours, minutes, and seconds, are converted into characters and are zero padded when appropriate. These zero-padded values are put together as a string with colons between them.

Climb_box is a routine which generates the vector VGD commands for the climb indicator box of the DATA PAGE. It is a very straight forward routine which consists of generating the five necessary connected line segments which form the climb indicator box. The only values needed by the routine are the plot coordinates for the box. The generated commands are stored a conversion array.

The **clim_hsh** routine is also a very straight forward routine. It generates the hash marks for the climb indicator of the DATA PAGE. These hash marks are just line segments plotted at certain intervals along the climb indicator arc. Four hash marks are generated by this routine at evenly spaced intervals. The intervals are determined by dividing a given arc into four parts. Note that the routine

43

does not put a hash mark in the middle of the arc. The information required by the routine consists of the plot coordinates of the climb indicator arc, the start and end angle of the climb indicator arc, and the radius of the climb indicator arc. The VGD commands generated by the routine are stored in a conversion array.

Clim_arrow is a routine which generates the VGD commands to plot the climb indicator arrow of the DATA PAGE. The routine generates this arrow given a plot location, angle, and radius. This arrow consists of three connected line segments. One corresponding to the shaft of the arrow and the other two making up the head of the arrow. The VGD commands for the arrow are stored in a conversion array.

Climb_rate is a function which retrieves the digital vertical speed value from the data package structure and converts this value using a linear conversion into feet per minute. This converted value is zero padded and is plotted as string at a given location on the vector graphics display. The clock package, alarm package, and plot coordinates are passed to the routine. The VGD commands generated are stored in a conversion array.

3.2.2 NAV PAGE Tools

The tools needed for the vector graphics development of the NAV PAGE vary greatly in their complexity. Some are

44

very straight forward and simple while others are rather complex and involve a large number of calculations and decisions to accomplish their tasks. The simple tools of this page consist of the following:

1) plane

2) waypoint

3) vortac

4) ndb

All of these simple routines basically the same. The first three involve connecting numerous line segments to form the appropriate graphic to be displayed. The ndb routine makes three calls of the aro_circ function which was discussed in the general tools section. The only information required by these routines is the plot coordinates for the graphic. All the VGD commands are stored in a conversion array.

The more complicated tools of the NAV PAGE consist of the following:

1) compass

2) heading_and_bearing

3) get_ndb_plot_angle

Compass is the routine which generates the vector graphics commands to plot the hash marks and appropriate markings for the compass portion of the NAV PAGE. A ninety degree portion of the compass is generated showing forty-

45

five degrees to either side of the plane's heading. The information which is required by this routine consists of the plot coordinates for center of the compass arc, the radius of the compass arc, and the heading of the plane. Hash marks are generated in intervals of five degrees. The hash marks which are at a compass headings which are divisible by ten are twice as long as those that are not divisible by ten. Markings along the compass are done at compass headings which are divisible by ten. These markings consist of headings between 0 and 360 degrees with markings of N, E, W, and S for respective headings of 0 degrees, 90 degrees, 180 degrees, and 270 degrees. The VGD commands generated by this routine are stored in a conversion array.

Heading_and_bearing calculates the heading and bearing of an NDB or VORTAC. It calculates these, given information consisting of the plane's heading in degrees and the plot angle of the VORTAC or the NDB. After calculating the heading and bearing the routine generates the appropriate VGD commands to plot the heading and bearing as strings on the VGD. The heading is plotted at distance two-thirds of the way between the NDB or VORTAC. The bearing is plotted at a distance one-third of the way between the NDB or VORTAC. The distance of the NDB or VORTAC is passed to the routine. Arrows are also drawn by the routine, with an arrow pointing "to" the NDB or VORTAC at the heading loca-

tion and an arrow pointing "from" the NDB or VORTAC at the bearing location. The VGD commands are stored in conversion array.

Get_ndb_plot_angle retrieves the digital value for the ADF from the data package and converts it into an appropriate NDB plot angle. It accomplishes this conversion by using a lookup table which is in the form of a structure. If the value retrieved does not equal a value in the lookup table, interpolation is used to get the appropriate plot angle for the NDB. The angle found is passed to the calling routine. The converted NDB plot angle is in degrees.

3.2.3 ILS PAGE Tools

Presently, there are only two page-specific tools on the ILS PAGE. The first tool is the **runway** function. **Runway** generates the necessary VGD commands to plot a runway on the display. The runway is plotted for a given location, scale factor, and angle. The plot coordinate is centered at the intersection to the runway centerline and touchdown line. The runway is drawn using vanishing-point techniques. The commands generated are stored in a conversion array.

Ils_compass is the second tool. This routine generates the vector graphics commands to plot the hash marks and appropriate markings for the compass portion of the ILS PAGE. A twenty-degree portion of the compass is generated

showing ten degrees to either side of the plane's heading. The information which is required by this routine consists of the plot coordinates for center of the compass arc, the radius of the compass arc, and the heading of the plane. Hash marks are generated in intervals of one degree. The hash marks which are at compass headings divisible by five are twice as long as the other hash marks generated. Markings along the compass are done at compass headings which are divisible by ten. These markings consist of headings between 0 and 360 degrees with markings of N, E, W, and S for respective headings of 0 degrees, 90 degrees, 180 degrees, and 270 degrees. The VGD commands generated by this routine are stored in a conversion array.

CHAPTER FOUR

HP 1345A VECTOR GRAPHICS

The HP 1345A is a high-resolution vector graphics dis-
play. It was chosen for the EHSI development system because
it is especially suited for this type of application. The
display is capable of operating up to 15,000 feet pressure
altitude. The plotting area is 9.5 cm high and 12.5 cm wide
and has 2048 addressable points in either direction. The
display also contains the vector memory option which allows
storage of up to 4096 command words and provides automatic
refresh capability for the CRT. The display also has a
built-in character set.

The HP 1345A accepts four basic commands:

1) SET CONDITION

2) PLOT

3) TEXT

4) GRAPH

These commands allow complete text and vector genera-

49

tion, while keeping programming overhead to a minimum. Note, each command word is 16 bits long. The GRAPH command was the only command which was not used in the development of the EHSI.

The SET CONDITION command is used to establish vector attributes. The attributes which it affects are line type, speed, and intensity. Using the line intensity and speed parameters, the user can generate up to 12 different intensity levels. This allows the user to intensify certain areas of the display and generate background areas. The vector generated will be the brightest when the line intensity is set at full brightness and the slowest writing speed. Once the SET CONDITION command has been executed, it stays in effect until another SET CONDITION command is executed. This command can be executed at any time. The bit patterns required for the SET CONDITION can be found in the found in Fig. 4.1[4].

The PLOT command is used to draw vectors on the display. The command moves a beam to a specific location in the Cartesian plane of the display each time an X-Y coordinate pair is received. The origin of the Cartesian plane is in the lower left hand corner of the display. The X-Y value at the origin is equal to (0,0). The plotting range is from 0 to 2047 in both the X and Y the directions. The beam can also be turned on and off by the command. A vector

is drawn from the previous beam coordinates to the coordinates specified by a new PLOT command. Note, all vectors are drawn in accordance with last SET CONDITION command received by the display. The bit patterns for the PLOT command can be found in Fig. 4.2[4].

An example will demonstrate how easy it is to use the PLOT command of the HP 1345A. Sending the following command words will draw a vector on the display. The command words are 16 bits wide and are shown in hexadecimal format.

COMMAND SENT	COMMAND WORD
1) SET CONDITION	$7818
2) PLOT X0(beam off)	$0000
3) PLOT Y0(beam off)	$0800
4) PLOT X1(beam off)	$00FF
5) PLOT Y1(beam on)	$18FF

Executing these commands will draw a line from the origin on the display to a point located at (255,255) on the display. The first command sets the line type to full solid, full brightness, and the slowest writing speed. The next two commands move the beam to the origin of the display with the beam off. The last two commands move the beam to (255,255) with beam on.

If a vector is to be drawn vertically, the Y value is the only command which is needed to establish the vector endpoint. This is possible because the display has a "last

51

X" register which stores the value of the last X location entered. This allows drawing of vertical vectors with one less command word.

There are a few limitations which should be noted when using the PLOT command of the HP 1345A. It was found that when drawing a vector, only one of the commands for the vector end location should be executed with the beam on. When both the commands for the end location were executed with the beam on, the display reacted unpredictably. The user should also take into account that the display width and height are different. But, there are 2048 addressable points in either direction. If this difference is not taken into account, a box will appear as a rectangle and a circle as an ellipse. To correct for this difference, a scaling factor must be used when calculating vector endpoints. The screen is 9.5 cm high and 12.5 cm wide; a scaling factor can be derived from this information. The user should also make sure that all vectors are plotted within the usable plotting area of the display. If vectors are drawn outside the vector drawing area, the vectors will be drawn to the opposite side of the display. This phenomenon is known as "wrap around". This can result in a very distorted image and should be avoided at all times.

The TEXT command is used to plot characters on the HP 1345A. The display has a complete built-in character set.

It is a modified ASCII set for graphics use. A portion of the set can be seen in Fig. 4.3[4]. The characters drawn on the display are drawn in accordance with the last SET CONDITION command received. The characters are always drawn at the slowest writing speed. Note, line type has no affect except on the largest character size. Characters are drawn at the last X-Y coordinates received.

Automatic spacing is provided between the characters when they are generated. Characters can be rotated up to four different ways and can be drawn in four different sizes. The starting position for each character is in the lower left hand corner of the defined character cell. Bit patterns for text generation can be found in Fig. 4.4[4].

The "wrap around" phenomenon which was discussed earlier applies to text generation as well. Characters should not be plotted to close to the plotting boundaries or "wrap around" will occur. If the boundary specifications shown in Fig. 4.5[4] are followed, no problems will be encountered when writing text at the screen boundaries.

The vector memory option of the HP 1345A makes the job of programming the display much easier. It allows the user to store up to 4096 command words and provides automatic refresh for the CRT at a 60 Hz rate or a rate defined by an external clock. This relieves the user processor of data storage and refresh requirements.

The vector memory recognizes two commands for programming. These commands are for memory address pointer manipulation and data transfer. Address-pointer operations are used for positioning data in the vector memory. Data transfer can be either a read or a write of data in the vector memory.

There are two pointers used to control access of data to and from the vector memory. The first pointer is not a user accessable pointer and is used for refresh purposes. The second pointer is the vector memory address pointer. This is the pointer which is used to control the part of the vector memory which is being accessed by the user. The bit patterns for the available vector memory commands can be seen in Fig. 4.6[4].

```
MSB                                                              LSB
D14 D13 D12 D11 D10 DO9 DO8 DO7 DO6 DO5 DO4 DO3 DO2 DO1 DO0
 1   1   I1  IO  X   X   L1  LO  O   X   W1  WO  X   X   X
```

Note: Bit 6 (DO6) must be zero.

Command Modifiers:

    a. To Set Line Intensity:

I1	IO	Intensity
O	O	Blank
O	1	Dim
1	O	Half Brightness
1	1	Full Brightness

    b. To Set Line Type:

L1	LO	Type
O	O	Solid Line
O	1	Intensified End Points
1	O	Long Dashes
1	1	Short Dashes

    c. To Set Writing Speed:

W1	WO	Speed
1	1	0.05 in. per microsec
1	O	0.10 in. per microsec
O	1	0.15 in. per microsec
O	O	0.20 in. per microsec

Figure 4.1 SET CONDITION COMMAND [4]

```
MSB                                                           LSB
D14 D13 D12 D11 D10 DO9 DO8 DO7 DO6 DO5 DO4 DO3 DO2 DO1 DOO
 O   O   XY  PC  D10 DO9 DO8 DO7 DO6 DO5 DO4 DO3 DO2 DO1 DOO
                 <------------------DATA------------------>
```

Command Modifiers:

    a. XY Information (D12)

        O = X coordinate (0-2047), specified by DOO-D1O
        1 = Y coordinate (0-2047), specified by DOO-D1O

    b. PC Beam Control Information (D11)

        O = Beam OFF (move)
        1 = Beam ON (draw)

Figure 4.2 PLOT COMMAND [4]

MOST SIGNIFICANT CHARACTER

	2	3	4	5	6	7
0	SP	0	@	P		p
1	!	1	A	Q	a	q
2	"	2	B	R	b	r
3	#	3	C	S	c	s
4	$	4	D	T	d	t
5	%	5	E	U	e	u
6	&	6	F	V	f	v
7		7	G	W	g	w
8	(	8	H	X	h	x
9	)	9	I	Y	i	y
A	*	:	J	Z	j	z
B	+	;	K	[	k	{
C	,	<	L	\	l	|
D	-	=	M	]	m	}
E	.	>	N	^	n	
F	/	?	O	_	o	

LEAST
SIGNIFICANT
CHARACTER

Figure 4.3   Partial 1345A MODIFIED ASCII CHARACTER
            SET [4]

```
MSB                                                          LSB
D14 D13 D12 D11 D10 DO9 DO8 DO7 DO6 DO5 DO4 DO3 DO2 DO1 DOO
 1   O   S1  SO  R1  RO  ES  DO7 DO6 DO5 DO4 DO3 DO2 DO1 DOO
                            CHARACTER--------------------->
```

Command Modifiers:

    a. ES Establish Size of Character

        O = Use previous size and rotation
        1 = Establish new size and rotation according to
            S1, SO, R1, and RO

    b. Rotate Character CCW

R1	RO	Rotation
O	O	O degrees
O	1	90 degrees
1	O	180 degrees
1	1	270 degrees

    c. Character Size

S1	SO	Size	W X H
O	O	1X	24 X 36
O	1	1.5X	36 X 54
1	O	2.0X	48 X 72
1	1	2.5X	60 X 90

Figure 4.4   TEXT COMMAND [4]

58

Figure 4.5 BOUNDARY LIMITS FOR CHARACTER PLOTTING [4]

VECTOR MEMORY WORD

```
MSB                                                      LSB
M15 M14 M13 M12 M11 M10 M9 M8 M7 M6 M5 M4 M3 M2 M1 M0
 0  B14 B13 B12 B11 B10 B9 B8 B7 B6 B5 B4 B3 B2 B1 B0
```

   Note: See data bit definitions for 1345A commands

INTERNAL JUMP

```
MSB                                                      LSB
M15 M14 M13 M12 M11 M10 M9 M8 M7 M6 M5 M4 M3 M2 M1 M0
 1   0   X   X  A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0
```

   Note: A11-A0 specify jump to vector memory address
         during refresh

ADDRESS POINTER

```
MSB                                                      LSB
M15 M14 M13 M12 M11 M10 M9 M8 M7 M6 M5 M4 M3 M2 M1 M0
 1   1   X   X  A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0
```

   Note: Sets address pointer to address specified by
         A11-A0

Figure 4.6   VECTOR MEMORY COMMANDS [4]

CHAPTER FIVE

VECTOR MEMORY LAYOUT FOR THE EHSI

Since the HP 1345A has the vector memory option, a
memory layout for the EHSI was designed. The vector memory
was broken up into seven separate areas. Three of these
areas contain the static data for the different pages of the
EHSI. The four other areas of the memory are used for the
dynamic portions of the EHSI. The fourth dynamic area is
used for the command line and warning messages which are to
be displayed.

The vector memory layout can be seen in Fig. 5.1. The
first word of the vector memory contains the jump vector
which determines which page of the EHSI is being displayed.
The jump vector is changed whenever one of the page keys is
pressed on the control keyboard. The first section of
memory contains the static information for the DATA PAGE.
This is located in the vector memory between $0001 and
$02FF. Note, the last command of the static data for the

DATA PAGE contains a jump to the location in the vector memory which contains the dynamic portion of the DATA PAGE. The second section of memory contains the static information for the NAV PAGE. This is located in the vector memory between $0300 and $04FF. Note, the last command of the static data for the NAV PAGE contains a jump to the location in the vector memory which contains the dynamic portion of the NAV PAGE. The third section of memory contains the static information for the ILS PAGE. This is located in the vector memory between $0500 and $06FF. Note, the last command of the static data for the ILS PAGE contains a jump to the location in the vector memory which contains the dynamic portion of the ILS PAGE. The fourth section of memory contains the dynamic information for the DATA PAGE. This is located in the vector memory between $0700 and $08FF. The fifth section of memory contains the dynamic information for the NAV PAGE. This is located in the vector memory between $0900 and $0BFF. The sixth section of memory contains the dynamic information for the ILS PAGE. This is located in the vector memory between $0C00 and $0DFF. Note, the last command of the dynamic data for all the pages of the EHSI contains a jump to the location in the vector memory which contains the command line and warning area for the EHSI. The final section of memory contains the command line and warning area for the EHSI. This is located in

vector memory between $0E00 and $0FFE. The final word of the vector memory contains a NO-OP. This NO-OP is necessary because a jump may not be made to another internal jump.

This memory layout can be changed at any time with the programming capability of the vector memory. This will make it possible to change the memory layout easily, when changes are made to the host-side of the EHSI program.

000H	Jump vector to static DATA, NAV, or ILS page.
001H	Static DATA page Last command is jump to 700H
	Not used
300H	Static NAV page Last command is jump to 900H
	Not used
500H	Static ILS page Last command is jump to C00H
	Not used
700H	Dynamic DATA page Last command is jump to E00H
	Not used
900H	Dynamic NAV page Last command is jump to E00H
	Not used
C00H	Dynamic ILS page Last command is jump to E00H
	Not Used
E00H	Command line screen code Last command is jump to FFFH
	Not used
FFFH	NO—OP

Figure 5.1   VECTOR MEMORY LAYOUT FOR THE EHSI

CHAPTER SIX

DATA PAGE

The DATA PAGE provides the pilot with basic flight information. It shows flight parameters which are inputed by the pilot prior to take-off or during the course of a flight and basic flight data showing the plane's status. Enabled alarm conditions are also shown on this page to alert the pilot as to which of the available alarms are set. For the most part, this page is in a digital type format. This means that most of the flight parameters are shown as digital values. The one portion of the DATA PAGE which is analog in appearance is the rate of climb indicator. This indicator will show the plane's rate of climb in feet per minute. An example of the DATA PAGE can be seen in Fig. 6.1.

The DATA PAGE is broken down into two basic parts. The first part consists of the static information for the page. This is information which will not change during the course

of the EHSI operation. This static information is loaded into the vector memory only once. This loading occurs when the EHSI system switch is turned "on". The second part consists of the dynamic information for the page which is generated during the course of the flight. When the DATA PAGE is the present page being displayed, this information gets loaded into the vector memory every time a clock interrupt is received from the DACI.

6.1 Static Portion

The static portion of the DATA PAGE can be broken down into three areas. The first area consists of text strings. These strings provide the headings for the information being displayed on the DATA PAGE. The strings are all generated using the (1.5) character size of the HP 1345A's built-in character set. All the plot locations for the strings are in an include file entitled "datpg_xy.h". This was done to allow the developer to easily change the location of the heading strings. This makes it possible to try new configurations on the DATA PAGE with a minimal amount of effort. Presently, some of the text strings are grouped together. Some of this grouping is random, while other groups of the strings share information which is similar in nature. The VGD commands used to plot the strings were generated using the string_gen routine which was discussed

in the tools chapter.   The text strings which are presently
shown on the DATA PAGE are the following:

1) HEADING: plane's present heading

2) AIRSPEED

    A) CAS: calculated airspeed

    B) TAS: true airspeed

3) GROUNDSPEED: plane's speed relative to the ground

4) ALTITUDES

    A) ASSIGNED: plane's assigned flight altitude

    B) PRESENT: plane's present altitude

    C) MDA/DH: minimum descent altitude/decision
           height

5) TIMER

6) TIME-OUT

7) FREQUENCIES

    A) COMMUNICATION: radio frequencies

    B) NAVIGATION: VORTAC frequencies

    C) ADF: automatic direction finder frequency

8) WAYPOINTS: intermediate navigation points on flight
    path

9) TEMPERATURE: outside temperature

10) BAROMETER: outside barametric pressure

11) REAL TIME VALUES

    A) MILITARY: military standard time

    B) EDT:

    C) ZULU: Greenwich Mean Time (GMT)

12) TIME SINCE LIFT-OFF

The second area of the static portion of the DATA PAGE is the climb indicator. The plot location for the indicator is also contained in the include file, "datpg_xy.h". Although the indicator is comprised of several different parts, the whole indicator can be moved by changing just one plot coordinate in the include file. This indicator is in the upper right corner of the display and consists of the following:

1) LETTERING: The letters are plotted vertically and spell out the word "CLIMB". The characters used are size (1.5) of the built-in character set. String_gen was used here.

2) INDICATOR ARC: The arc was plotted using arc_circ. It provides the boundary for the meter.

3) INDICATOR CENTERLINES: Line was used to plot these. They provide the zero climb rate reference for the meter.

4) CLIMB BOX: The climb_box routine was used here. The climb box provides a starting point for the climb indicator arrow which is in the dynamic part of the page. It also boxes in the indicator's lettering.

5) INDICATOR HASH MARKS: These marks provide reference points on the indicator at certain climb rates. These reference points are at rates of -2000, -1000, +1000, and +2000 feet per second. They are plotted using clim_hsh.

The third area contains the command line. Below this line is where all messages to the pilot will be displayed. These include both warning and error messages. This area

68

will show numeric entries from the control keyboard and results of operations performed with the HP-style calculator functions available on the EHSI. This line is plotted using the **line** routine.

6.2 Dynamic Portion

The dynamic portion of the DATA PAGE is responsible for plotting the flight status information, flight parameters entered by the pilot, and the climb indicator arrow of the climb indicator. This portion also uses the "datpg_xy.h" coordinate file when plotting the vector graphics which it generates. Note, all the VGD commands used to plot strings in this portion of the page are generated using **string_gen**. Also, all zero padding of values is done with the **zero_pad** routine. The dynamic section of this page consists of the following:

1) HEADING: The heading of the plane is calculated using the **get_heading** routine. It is then zero padded and plotted.

2) AIRSPEED: The airspeed of the plane is calculated using the **airspeed** routine. It is then zero padded and plotted. Alarm conditions are also monitored by this routine for the stall airspeed.

3) GROUNDSPEED: The groundspeed has yet to be implemented.

4) ALTITUDES

   A) ASSIGNED ALTITUDE: The assigned altitude is retrieved from the clock package, zero padded, and plotted. This is entered by the pilot.

69

B) PRESENT ALTITUDE: The present altitude is
   calculated using the **altitude** routine. It is
   then zero padded and plotted. Alarm con-
   ditions are monitored for both assigned alti-
   tude and MDA/DH.

C) MDA/DH: The MDA/DH is retrieved from the clock
   package, zero padded, and plotted. This is
   inputed by the pilot.

5) TIMER and TIME-OUT: The **timer** routine is called
   for both these values. The timer routine calc-
   ulates the TIMER value and monitors the alarm con-
   dition for the TIME-OUT value. The TIME-OUT value
   is entered by the pilot. Both values are zero
   padded and plotted.

6) FREQUENCIES

   A) COMMUNICATION: The communication frequencies
      are retrieved from the clock package, zero
      padded, and plotted. These are entered by
      the pilot.

   B) NAVIGATION: The navigation frequencies are re-
      trieved from the clock package, zero padded,
      and plotted. These are entered by the pilot.

   C) ADF: The ADF frequency is retrieved from the
      clock package, zero padded, and plotted. It
      is entered the pilot.

7) WAYPOINTS: Waypoints have yet to be implemented.
   They will be entered by the pilot.

8) TEMPERATURE: Temperature has yet to be implemented.
   It will be entered by the pilot and will be dis-
   played in both fahrenheit and celsius.

9) BAROMETER: The barometer has yet to be implemented.
   This will be entered by the pilot.

10) REAL-TIME CLOCK:

    A) MILITARY: The **time_string_gen** routine is
       called which retrieves the clock values, zero
       pads them, and generates the time string.
       The time string is then plotted.
    B) EDT: EDT time has yet to be implemented.

70

C) ZULU: ZULU time has yet to be implemented.

11) TIME SINCE LIFT-OFF: This has yet to be
    implemented.
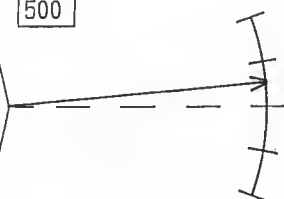
Figure 6.1   PRESENT FLIGHT DATA PAGE

6.3 High Level Flowchart

```
      ( START )                    │                              │
          │                        ▼                              ▼
          ▼                 ┌──────────────┐             ┌──────────────┐
  ┌──────────────┐          │ Put up       │             │ Put up       │
  │ Get the      │          │ MDA/DH       │             │ COM1         │
  │ DATA         │          └──────────────┘             └──────────────┘
  │ PACKAGE      │                 │                              │
  └──────────────┘                 ▼                              ▼
          │                 ┌──────────────┐             ┌──────────────┐
          ▼                 │ Put up       │             │ Put up       │
  ┌──────────────┐          │ rate of      │             │ COM2         │
  │ Put up       │          │ climb        │             └──────────────┘
  │ REAL TIME    │          └──────────────┘                    │
  └──────────────┘                 │                              ▼
          │                        ▼                       ┌──────────────┐
          ▼                 ┌──────────────┐             │ Put up       │
  ┌──────────────┐          │ Put up       │             │ VOR1         │
  │ Put up       │          │ rate of      │             └──────────────┘
  │ plane's      │          │ climb        │                    │
  │ HEADING      │          │ arrow        │                    ▼
  └──────────────┘          └──────────────┘             ┌──────────────┐
          │                        │                       │ Put up       │
          ▼                        ▼                       │ VOR2         │
  ┌──────────────┐          ┌──────────────┐             └──────────────┘
  │ Put up       │          │ Put up       │                    │
  │ ASSIGNED     │          │ plane's      │                    ▼
  │ ALTITUDE     │          │ AIRSPEED     │              ┌──────────────┐
  └──────────────┘          └──────────────┘             │ Put up       │
          │                        │                       │ TIMER and    │
          ▼                        ▼                       │ TIME-OUT     │
  ┌──────────────┐          ┌──────────────┐             └──────────────┘
  │ Put up       │          │ Put up       │                    │
  │ plane's      │          │ ADF          │                    ▼
  │ ALTITUDE     │          └──────────────┘             ┌──────────────┐
  └──────────────┘                                         │ Send data    │
                                                           │ to vector    │
                                                           │ memory       │
                                                           └──────────────┘
                                                                  │
                                                                  ▼
                                                            ( RETURN )
```
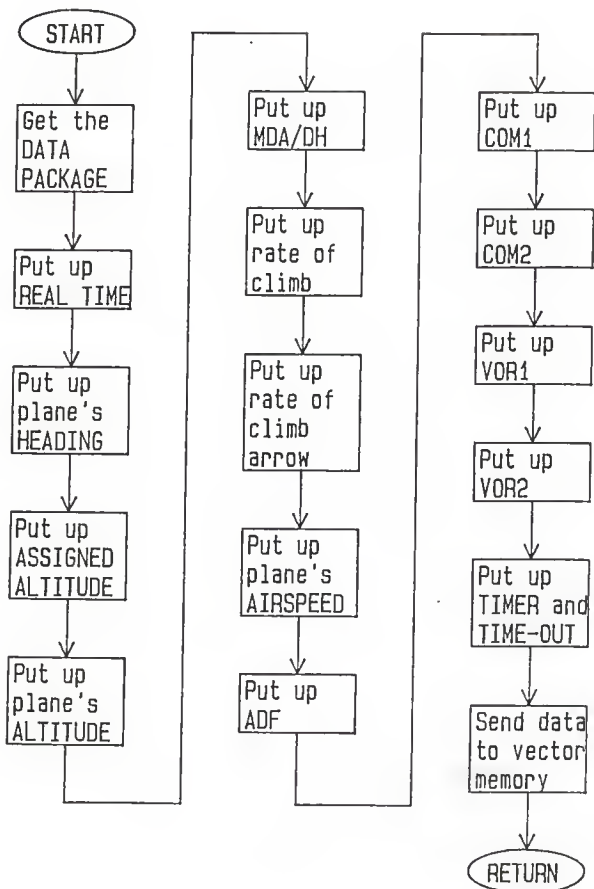
CHAPTER SEVEN

NAV PAGE

The NAV PAGE is probably the page which is of most interest to the small plane pilot. It provides a large of amount of information to the pilot. This information is assimilated from various flight instruments and is displayed in a format which will show the position of the plane rela- tive to known navigational fixes. Unlike the DATA PAGE, most of the information on this page is provided in an analog type format. An example of the NAV PAGE can be seen in Fig. 7.1. The information supplied on this page consists of the following:

1) HEADING:

    A) The heading is displayed in an analog type compass format.

    B) A digital value for the heading is displayed at the compass heading.

2) VORTAC:

    A) The heading "to" and bearing "from" the VORTAC are displayed along with appropriate

74

arrows.

B) DME information to the VORTAC is displayed.

C) The VORTAC is displayed in its position relative to the plane.

3) NDB:
   A) The heading "to" and bearing "from" the NDB are displayed along with appropriate arrows.

   B) Later, possibly, distance information to the NDB will be implemented.

   C) The NDB is displayed in its position relative to the plane.

4) MISCELLANEOUS INFORMATION: This information is displayed in a digital format.

   A) AIRSPEED

   B) GROUNDSPEED

   C) TRACK

   D) ALTITUDE

   E) DME

5) WARNING MESSAGE AREA: Warning messages to the pilot are displayed in this area. These messages are generated by various routines.

6) COMMAND LINE AREA: This area is for displaying numeric values entered by the pilot via the control keyboard.

This page will make it much easier for the pilot to navigate an established course between two points. Rather than having to draw a mental picture of the plane's position relative to known navigational fixes, this position will be shown graphically with all relative navigational fixes displayed.

The programming for the NAV PAGE is broken down into two basic parts as in the DATA PAGE. The first part consists of the static information for the page. The second part consists of the dynamic information. Remember, the static information is loaded into the vector memory only once when the EHSI system switch is turned "on". When the NAV PAGE is the present page being displayed, the dynamic information is generated and loaded into the vector memory every time a clock interrupt is received from the DACI.

7.1 Static Portion

The static portion of the NAV PAGE can be broken down into four areas. The first area consists of text strings. These strings provide the headings for the information being displayed on the NAV PAGE. All the plot locations for the strings are in an include file entitled "navpg_xy.h". This is the same type of include file that was used in the DATA PAGE for its plot locations. All the strings are generated using the (1.5) character size of the built-in character set. **String_gen** was used to generate the VGD commands to plot the strings. The text strings which are presently shown on the NAV PAGE are the following:

1) AIRSPEED: calculated airspeed

2) GROUNDSPEED: speed relative to ground

3) TRACK: ground track of airplane

Figure 7.1   PRESENT NAVIGATIONAL PAGE
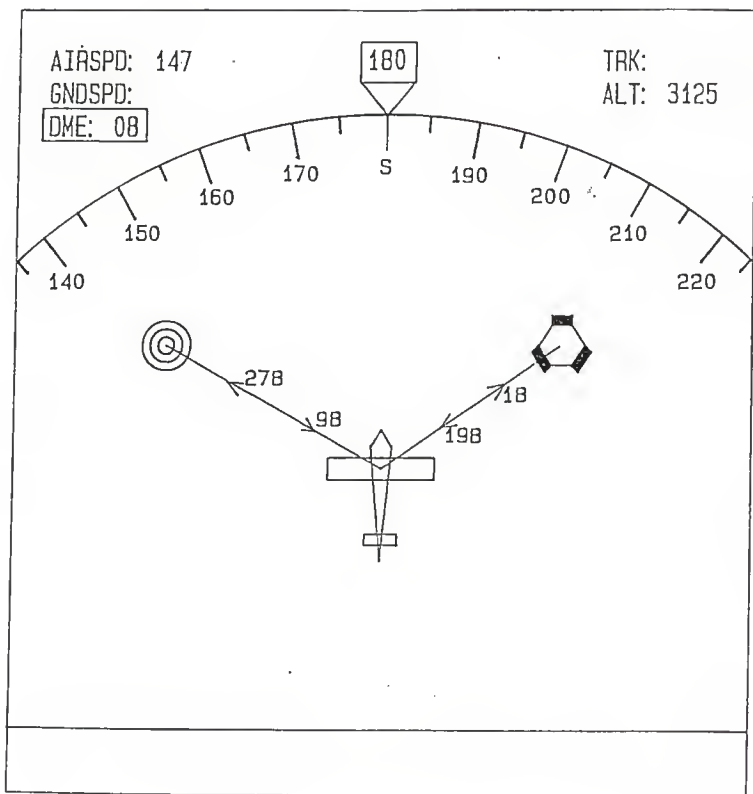
4) ALTITUDE: present altitude

5) DME: distance information to VORTAC or localizer

The second area of the static portion is the plane graphic. The plot location for the plane is also contained in the "navpg_xy.h" file. It is plotted in the center of the NAV PAGE and is generated using the plane routine. All VOR's and NDB's are plotted relative to this plane graphic. Therefore, this graphic represents the position of the plane relative to the navigational fixes being plotted.

The arc of the analog compass generated by the NAV PAGE is the third area of this portion. This arc has an angle of ninety degrees and is plotted at the top of the NAV PAGE. Its plot location is contained in the plot coordinate in- clude file. All the static text strings generated are displayed above this arc.

The fourth area of the static portion of the NAV PAGE is the heading box and pointer. This area will contain the digital heading of the airplane and will provide a pointer to the plane's heading on the compass. It is plotted above the compass arc in the middle of the display. Its plot coordinates are in the plot coordinate include file.

7.2 Dynamic Portion

The dynamic portion of the NAV PAGE is much more comp- licated than that of the DATA PAGE. It is responsible for

calculating and plotting the flight status information men-
tioned and generating the graphical representation of the
plane's position relative to the known navigational fixes.
The latter of these is the more complicated part of this
portion. It involves assimilating all the flight infor-
mation about the known navigational fixes collected by the
DACI and generating the VGD commands to plot this assimilat-
ed data in a usable graphical representation. All plot
coordinates for the vector graphics generated in this
portion are also contained in the "navpg_xy.h" include file.
All strings are plotted using **string_gen** and are zero padded
using **zero_pad**. The dynamic portion of the NAV PAGE con-
sists of the following:

1) AIRSPEED: The airspeed of the plane is calculated
   using the **airspeed** routine. It is then zero
   padded and plotted. Alarm conditions are also
   monitored by this routine for the stall airspeed.

2) GROUNDSPEED: The groundspeed has yet to be im-
   plemented.

3) TRACK: The track has yet to be implemented.

4) ALTITUDE: The present altitude is calculated using
   the **altitude** routine. It is then zero padded and
   plotted. Alarm conditions are monitored for both
   assigned altitude and MDA/DH.

5) DME: The DME value is calculated using the **dme**
   routine. It is then zero padded and plotted.

6) HEADING

   A) The heading is calculated using the **get_heading**
      routine. It is then zero padded and plotted in
      the heading box.
   B) The **compass** routine is then used to generate

the dynamic portion of the compass.   The head-
ing is passed to the routine.   This routine
then generates the vector graphic commands to
put the hash marks and markings for the heading
sent.   The heading of the plane is centered at
the base of the heading pointer below the head-
ing box.

7) VORTAC

    A) A VORTAC is displayed if it is within 20 naut-
       ical miles.

    B) Position of VORTAC relative to plane is calc-
       ulated.

    C) VORTAC's angle relative to plane is calculated.

    D) Heading information is added into this angle
       and the angle range is checked.

    E) Distance to VORTAC is scaled.

    F) VORTAC is plotted using the **vortao** routine.

    G) If the plot distance of the VORTAC is greater
       than a certain limit the heading "to" and the
       bearing "from" the VORTAC are generated.

        1. **Heading_and_bearing** is called to calc-
          ulate the heading and bearing.

        2. The heading and bearing are plotted with
          appropriate arrows.

8) NDB

    A) The angle of the NDB relative to the plane is
       calculated using the **get_ndb_plot_angle** rou-
       tine.

    B) The NDB is then plotted at a constant distance
       from the plane graphic using the **ndb** routine.

    C) **Heading_and_bearing** is called to calculate the
       heading and bearing.

    D) The heading and bearing are plotted with
       arrows.

7.3 Future Development

The present status of the NAV PAGE leaves allot of room for improvement. There are many things which could be added to the page and several problems which must be overcome.

The first thing which must be added to the page is the use of waypoints. These waypoints will provide the pilot with intermediate navigational fixes. A routine must be written which allow the pilot to enter the waypoints. The dynamic portion of the NAV PAGE will also need to be modified to incorporate the calculation of the waypoints position relative to the airplane.

Second, the ground track will need to be implemented. This will involve writing a routine to calculate the ground track. Another routine will also be required to plot the ground track arrow above the compass arc.

Third, a wind speed vector will need to be added. This will involve using basic vector mechanics and the available flight information provided by the DACI after it has been converted.

Fourth, an additional vortac will have to be implemented. This will be necessary because most small planes can have two VORTAC's tuned in at once. This addition will involve slight modifications of the dynamic portion of the NAV PAGE to incorporate the navigational frequencies entered by the pilot. These frequencies will be used by the

routine, along with the information being used presently, to calculate the position of each of the VORTAC's relative to the plane.

Fifth, a data base should be added. This data base could contain the navigational coordinates for VORTACS's, NDB's, area airports, cities, towns, and other known landmarks. Such information would be invaluable when calculating the position of waypoints. It would also make it possible to show the position of the plane relative to navigational fixes in the data base.

Sixth, a way of calculating the distance to an NDB would be convenient. If a data base existed this calculation would be easy. Without a data base, however, another method must be found to calculate this distance. A possible solution is found in Lagerberg's thesis[2].

The first problem which must be overcome is that of processing speed. Presently, the dynamic portion of the NAV PAGE takes the most time of all the routines. However, this may not prove to be true when the ILS page is developed further. A way must be found to speed up the VGD update as close to real time as possible. This will probably involve moving to a faster processor and coprocessor along with a decoded and latched keyboard.

Second, there is a problem the positional information for the VORTAC's obtained from the flight simulator. The

positional information retrieved from the flight simulator is the form of rectangular coordinates. Thus, the position of the plane can viewed as the origin of a cartesian coordinate system with the VORTAC located at some specific coordinate in that system. Positional information to a VORTAC in a real plane is in the form of radial information along with DME information. Therefore, minor changes will have to be made in the dynamic portion of the software to calculate the position of VORTAC's relative to the plane.

7.4 High Level Flowchart

START

Get the
DATA
PACKAGE

Put up
plane's
AIRSPEED

Put up
GROUND
SPD & TRK

Put up
plane's
ALTITUDE

Put up
DME

Put up
plane's
HEADING

Put up
Compass

DME
<=30
NM?   N → 2

Y

Calculate
DELTA_X &
DELTA_Y
of VOR

Calculate
angle
of VOR
using
DELTA_X,
DELTA_Y,
& plane's
HEADING

Calculate
VOR plot
radius

Calculate
VOR plot
coord.

Put up
VOR

1
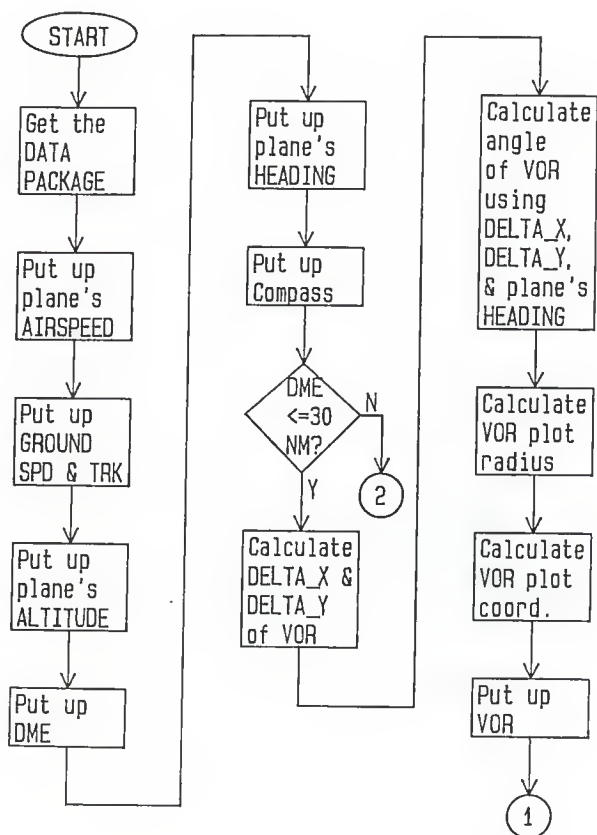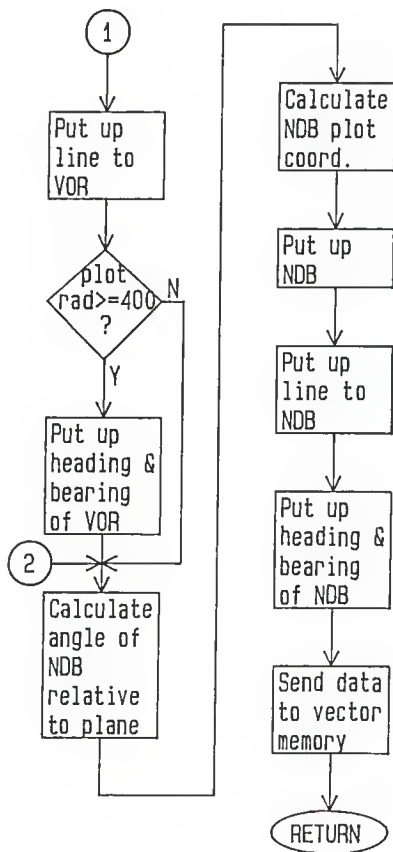
CHAPTER EIGHT

ILS PAGE

The ILS PAGE provides a graphical representation of an ILS approach to an airfield. This is something which will be of great interest to the small plane pilot who has an IFR rating or who is in the process of obtaining one. A large amount of information is assimilated from the flight instruments on this page. This assimilated information is presented in a format which will show the plane's position relative to the approach path. This will make it easier for a pilot to stay on the approach path. The pilot will no longer have to rely on a mental picture of where the plane is located relative to the approach path. At present, a very simple graphical representation of an ILS approach is being generated. However, future development of this page will result in a very complex graphical representation which will make it even easier for the pilot to fly an ILS approach. The algorithm for the future ILS PAGE will be

much more complicated and involve a larger amount of calculations than the present algorithm. The ILS PAGE will then be the most complicated of the three pages of the EHSI.

Most of the information on this page is in an analog type format. An example of the present version of the ILS PAGE can be seen in Fig. 8.1. The information presented on this page is the following:

1) HEADING:

    A) The heading is displayed in analog type compass format showing a 10 degree view either side of the heading.

    B) A digital value for the heading is displayed at the compass heading.

2) ALTITUDE:

    A) PRESENT: The present altitude of the plane is displayed in a digital format.

    B) MDA/DH: The MDA/DH is displayed in a digital format.

3) DME: The DME information to a localizer or VORTAC is displayed in a digital format.

4) APPROACH STATUS: This is in an analog format.

    A) Plane's position relative to glideslope is displayed.

    B) Plane's position relative to localizer is displayed.

    C) Scaled runway is shown.

    D) Heading of runway relative to the plane's heading is shown.
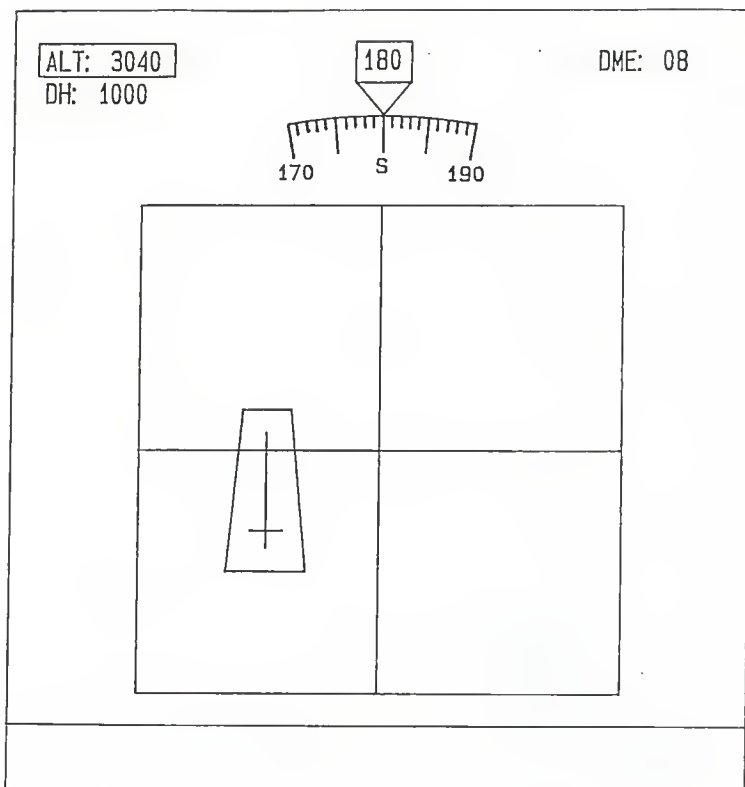
Figure 8.1 PRESENT INSTRUMENT LANDING SYSTEM (ILS) PAGE

5) WARNING MESSAGE AREA: Warning messages to the pilot
   are displayed in this area.   These messages are
   generated by various routines.

6) COMMAND LINE AREA: This area is for displaying num-
   eric values entered by the pilot via the control
   keyboard.

The programming for the ILS PAGE is also broken up into
two basic parts just like the DATA and NAV pages.   The first
part   generates   the static information for the   page.   The
second part generates the dynamic information for the   page.
Once again, the static information is loaded into the vector
memory only one time.   When the ILS PAGE is the present page
being displayed,   the dynamic information is generated and
loaded   into the vector memory every time a clock   interrupt
is received from the DACI.

8.1 Static Portion

The   static   portion of the ILS PAGE consists of three
separate   areas.   The   first   area   is   comprised   of   text
strings.   These   strings   provide headings for   the   flight
status information being displayed on the page.   This   page
also   has an include file of the same type used in the   DATA
and   NAV   pages which contains the plot coordinates for   the
strings.   This   file   is entitled   "ilspg_xy.h".   All   the
strings are generated using the (1.5) character size of   the
HP 1345A internal character set.   String_gen is used to plot
the strings.   The text strings presently being displayed on

the ILS PAGE are the following:

1) ALTITUDE: present altitude

2) MDA/DH: minimum decision altitude or decision height

3) DME: distance information to VORTAC or localizer

The second area of this portion of the page consists of the arc of the analog compass. This is a twenty degree arc which is plotted at the middle of the top part of the page. It is generated with the **arc_circ** routine and its plot coordinates are contained in the coordinate include file.

The third area of the static portion of the ILS PAGE consists of the heading box and pointer. This area will contain the digital heading of the airplane and will provide a pointer to the plane's heading on the twenty degree portion of the analog compass. Its plot coordinates are also in the coordinate include file. It is plotted above the compass arc in the middle of the display.

8.2 Dynamic Portion

The dynamic portion of the ILS PAGE generates the graphical representation of the plane's position relative to the approach path. It also plots the flight status information which was mentioned above. To generate the representation of the approach, data is assimilated from the localizer, glidepath, heading, and DME. As was stated pre-

viously, the present version is a very simple. This is the second most complicated page of the EHSI. All plot coordinates for the vector graphics generated by this portion are in the "ilspg_xy.h" include file. All strings are plotted using **string_gen** and are zero padded using the **zero_pad** routine. The dynamic portion of the ILS PAGE consists of the following:

1) ALTITUDE: The present altitude is calculated using the **altitude** routine. It is then zero padded and plotted. Alarm conditions are monitored for both assigned altitude and MDA/DH.

2) MDA/DH: The MDA/DH is retrieved from the clock package, zero padded, and plotted. This is inputed by the pilot.

3) HEADING

A) The heading is calculated using the **get_heading** routine. It is then zero padded and plotted inside the heading box.

B) The **ils_cmps** routine is then used to generate the dynamic portion of the compass. Remember, this compass only shows a twenty degree portion around the heading. The heading of the plane is centered at the base of the heading pointer below the heading box.

4) DME: The DME value is calculated using the **dme** routine. It is then zero padded and plotted.

5) APPROACH STATUS

A) If plane is within 20 nautical miles, the following procedure is executed.

1. The runway scale is calculated.

2. The plot boundaries are calculated based on the runway scale.

91

3. The crosshairs are plotted.

4. If the plane is within 15 nautical miles, the following procedure is executed.

    a. The approach sensitivity is calculated based on the DME value.

    b. The glideslope is calculated using the approach sensitivity and boundary conditions.

    c. The course deviation from the localizer is calculated using the approach sensitivity and boundary conditions.

    d. The crab angle is calculated using the plane heading and the runway heading.

    e. The runway is then plotted based on the glideslope, course deviation, and runway scale.

8.3 Future Developments

There are many improvements which could be made to the present version of the ILS PAGE. Eventually, these improvements will probably result in a graphical representation of an ILS approach which is only vaguely similar to the present version. An example of what the future ILS PAGE might look like appears in Fig. 8.2 and Fig. 8.3[2].

The later versions of software which generate the ILS PAGE will assimilate a larger amount of data from the flight instruments than the present version of the page. In the final version of the EHSI, the ILS PAGE will become the most complicated of the three pages available on the system.
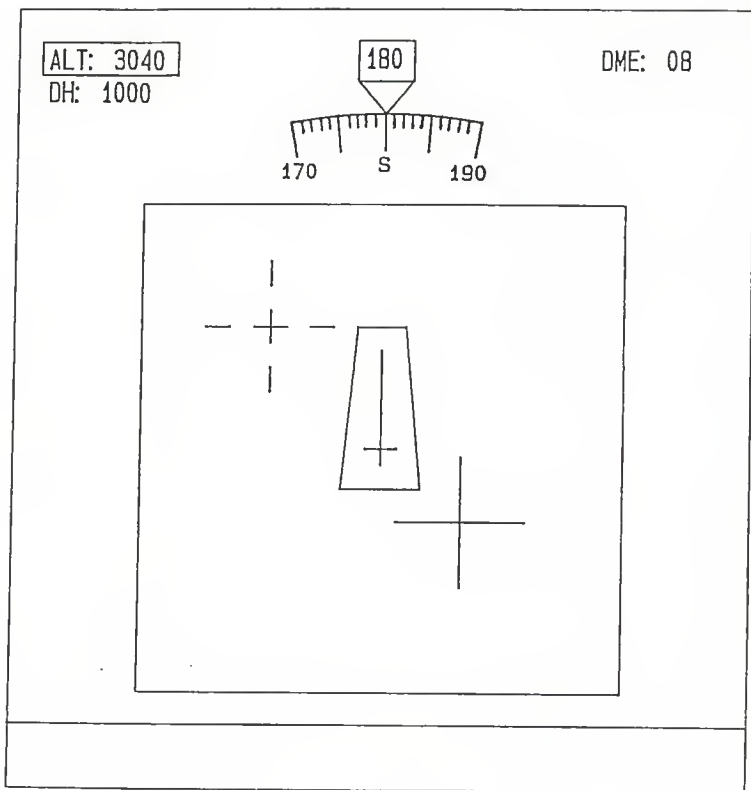
Figure 8.2   ILS PAGE USING TWO CROSSHAIRS

93
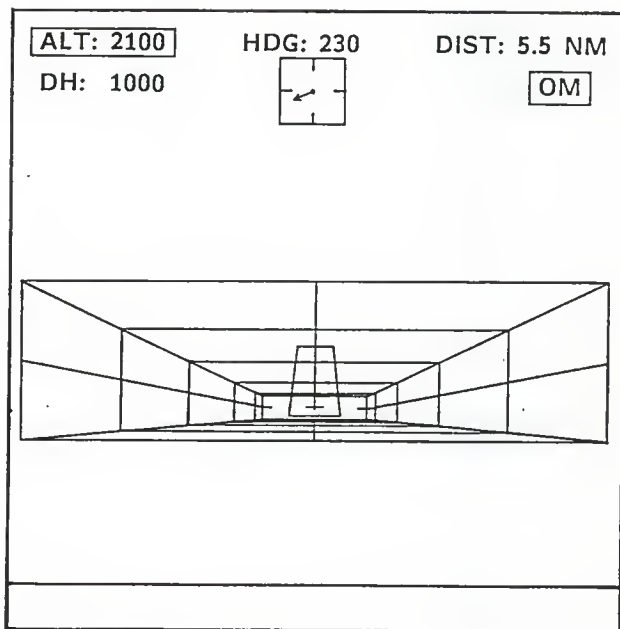
Figure 8.3   ILS PAGE WITH APPROACH TUNNEL [2]

94

The first thing which might be added to the page is the addition of tendency information. This information will be very important to the pilot. It will give immediate feed-back about whether or not a maneuver being made is going to get the plane back on the approach path. Digital filters will have to be written to accomplish this task. Gruen-bacher discusses the development of such filters in his thesis [3]. An example of tendency representation can be seen in the box with the arrow at the top of the display in Fig. 8.3[2]. The arrow represents the plane's tendency to move off the glideslope and localizer. If the arrow is pointing up and to the right, the plane's tendency is to move up and away from the approach path. Another example of representing the tendency information can be seen in Fig. 8.2. The dotted crosshairs and solid crosshairs are used to indicate the plane's tendency. When both the crosshairs are on top of one another the plane is on the approach path.

Second, an ILS approach tunnel might be added. This tunnel would display the boundaries of the approach path. The outer and middle marker information would be represented by the beginning and end of the tunnel. The plane's posi-tion on the approach path would be indicated by the graph-ical representation of the tunnel on the VGD. An example of the ILS tunnel can be seen in Fig. 8.3[2].

Third, outer and middle marker information needs to be

incorporated into the ILS page. This was mentioned in the previous paragraph. This information is important because it lets the pilot know how far he is into an approach. There are many possible ways the marker information might be presented.

Fourth, the addition of a data base like the one discussed in the future developments of the NAV PAGE would be of great importance. This data base could contain information for each airport such as the runway heading, outer marker distance, middle marker distance, MDA/DH, and field altitude. This information would be invaluable in generating any version of the ILS PAGE.

Fifth, reverse course information would be a convenient addition. This would allow the pilot to fly outbound on an ILS approach path without having to change his reactions to the graphical representation of the approach. This would be especially nice on the present version of the ILS PAGE. If an ILS tunnel was being used, this information would not would not be necessary.
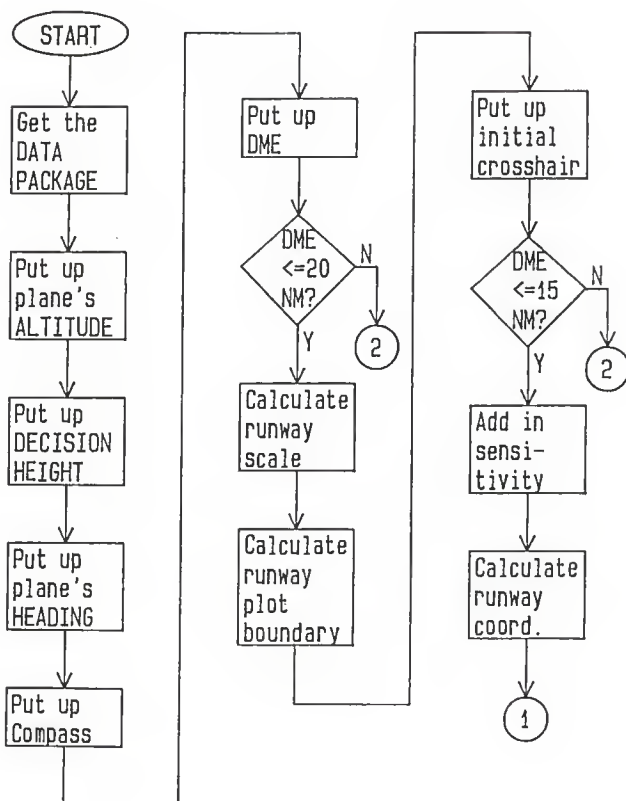
8.4 High-level Flowchart
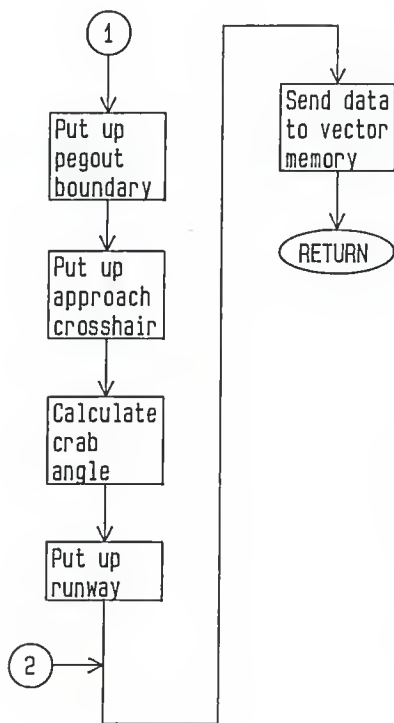
CHAPTER NINE

CONCLUSION

9.1 Results

The present version of the EHSI is still basically in its infancy. However, a large amount of progress has been made since the initial proposal by Dyer in 1982[1]. An EHSI Development System now exists, so that the different pages of the EHSI can be developed. This development system is discussed briefly in Chapter 1 and in detail in [2]. Usable communication routines have been developed to allow the host computer to communicate with the DACI of the development system. These routines are discussed in Chapter 1 and in detail in [3]. The initial software to generate the graphics for the three pages of the EHSI has been developed. This software is written in the C programming language and is discussed Chapters 2, 6, 7, and 8. The code for the necessary routines can be seen in Appendix E. The DATA PAGE and the NAVIGATIONAL PAGE are the pages which have seen the

most progress. The ILS PAGE is going to require much more development.

The present working version of the EHSI gives confidence that the goal of providing an affordable EHSI for the general-aviation public can be reached. However, the final version of the EHSI will take several more person-years of development.

9.2 Future Development

Along with the future development of the NAVIGATIONAL PAGE and the ILS PAGE discussed in Chapters 7 and 8, there are several other areas which must be developed. First, EEPROM's must be programmed with the DACI control program. This will include the code in [2] and the additions discussed in Appendix A.

Second, the possibility of developing a dedicated graphics processor should be considered. This would take the burden of generating the VGD commands to produce the various graphics of the EHSI pages off the host. The host could then be used for evaluating DACI interrupts, making decisions, and processing data retrieved by the DACI.

Third, a faster processor and coprocessor are needed on the host side of the development system. This could be accomplished by upgrading the present host to a faster PC with a faster numerical coprocessor. Or, a dedicated host

processor could be designed to handle the host-side respons-
ibilities of the development. This is probably the most
logical way to proceed after the pages have been fully
developed because the final EHSI system will have to contain
such a dedicated processor.

Fourth, a data base should be incorporated into the
EHSI system. This would be valuable in the development of
all the pages of the EHSI. The benefits of data base are
discussed in the future development sections of Chapters 7
and 8. Such a data base will have to be implemented in the
final version of the EHSI to make it a highly marketable
product.

9.3 Another Application

As was stated, the goal of our EHSI is to provide an
affordable EHSI for the general-aviation public. This EHSI
will ease the burden of the single pilot flying under IFR
conditions. The EHSI will be designed for use in the small
airplane. However, this will not be its only application.
The EHSI would be invaluable in a teaching environment. It
could be used to help a pilot obtaining an IFR rating to
better understand what the instruments are telling him.
This will be especially true on the NAVIGATIONAL and ILS
PAGES of the EHSI. Oftentimes it is difficult to draw a
mental picture of the plane's position relative to a navi-

gational fix or to get immediate feedback as to whether or
not a maneuver will get the plane back onto a selected
course or ILS approach path.   The EHSI will provide this
information for the pilot and speed up the learning process.

REFERENCES

[1]   Dyer, S.A., "A Proposed Electronic Horizontal
        Situation Indicator for use in General-Aviation
        Aircraft," Proceedings of 1982 Position, Location
        and Navigation Symposium, pp. 198-205.

[2]   Lagerberg, J.D., An Electronic Horizontal Situa-
        tion Indicator and Development System, Kansas
        State University, 1987.

[3]   Gruenbacher, D.J., Low-level Software for an EHSI
        Development System, Kansas State University, 1987.

[4]   Hewlett-Packard Inc., HP-1345A Digital Display
        Module Designers Manual, 1981.

[5]   UnderWare Inc., BRIEF Quick Reference Card Version
        1.33, 1984.

APPENDIX A

CHANGES TO THE DACI

First, dedicated interrupt lines were added to the DACI. This involved both hardware and software modifications to the DACI. This addition greatly improved the reliability of the system. Prior to this addition, noise on the communications bus was causing the DACI to interpret noise spikes as interrupt requests from the host. Since there was no interface command on the bus at the time of the interrupt, the interrupt was determined to be an illegal interface command and an error message resulted from the DACI.

The hardware changes to implement the dedicated interrupt lines can be seen in Fig. A.1[2] and Fig. A.2[2]. These dedicated lines are called IRQOUT and IRQIN. The IRQOUT line is used when the DACI wishes to interrupt the host. The IRQIN line is used when the DACI is being interrupted by the host. In Fig. A.1, the original OUTSHAKE line

104

of the DACI was moved from pin 19 of the EHSI parallel port connector to pin 25 of the parallel port connector. The dedicated IRQOUT line was then connected to pin 19 of the parallel port. The IRQOUT line originates from pin 14 of the control PIA in Fig. A.2. The SPAREIRQ line was originally connected to the control PIA at pin 14 and was not being used. The dedicated IRQIN line was implemented by cutting the original INSHAKE line going to pin 5 of U13 in Fig. A.2 and adding a 4.7k ohm resistor as a pullup. This IRQIN line is connected at pin 6 of the EHSI parallel port connector as seen in Fig. A.1. Pins 4, 23, 21, 26 of the parallel port were also grounded.

The software additions for the dedicated interrupt scheme were very simple. Only one additional routine was needed and it is entitled OUTSHAKE_IRQ. This routine is presently located at $28DC in the RAM of the DACI. The code for OUTSHAKE_IRQ is seen in Fig. A.3. Additional changes to other DACI routines were also necessary to incorporate the OUTSHAKE_IRQ routine. Copies of the original DACI routines from [2] are at the end of this appendix. The changes made are noted in these routines. Note, PB4 of the control PIA in Fig. 2.A is configured as an output for the OUTIRQ line.

Second, a switch was added to the DACI to allow the input of an external clock to the clock interrupt line of the DACI. This was added to analyze the speed of the EHSI

system and to demonstrate the operation of the interrupt stack on the host-side. It also made it possible to retrieve data at specific frequencies for the design of digital filters for various portions of the EHSI. The hardware modifications that were made are seen in Fig. A.4. The CLKIRQ line was cut and a two-pole switch was added. One pole of the switch is connects the CLKIRQ line to the real-time clock interrupt of the system. The other pole connects the CLKIRQ line to an external clock input.

Figure A.1   DACI TO HOST PARALLEL PORT [2]
ADDITION OF IRQIN AND IRQOUT

107

Figure A.2   CONTROL PIA AND ALARM HORN [2]
ADDITION OF IRQIN AND IRQOUT

108

```
*****************************************************************

                    SUBROUTINE OUTSHAKE_IRQ

PURPOSE:            To send out a pulse on the IRQOUT line
                    of the parallel port.

INITIAL CONDITIONS: This subroutine requires no parameter
                    to be passed to it.

ACTION:             The pulse on the IRQOUT line has a pulse
                    width of approximately 25 microseconds.
                    The actual pulse width can be varied
                    according to the value in D0.L.  The
                    expression for the pulse width is

                        3 * (D0 + 1) + 4 microseconds.

REGISTER USAGE:     No registers are affected.

*****************************************************************

        ORG.      $208DC
        MOVE.L    D0,$9C8         SAVE REGISTER D0
        MOVE.L    #6,D0          SET PULSE LOW WIDTH
        BCLR      #4,$30005      BRING OUTIRQ LOW
SELF5   DBRA      D0,SELF5       HOLD OUTIRQ LOW
        BSET      #4,$30005      TAKE OUTIRQ HIGH
        MOVE.L    $98C,D0        RESTORE REGISTER D0
        RTS                      RETURN TO CALLING ROUTINE
```

Figure A.3 OUTSHAKE_IRQ ROUTINE

109
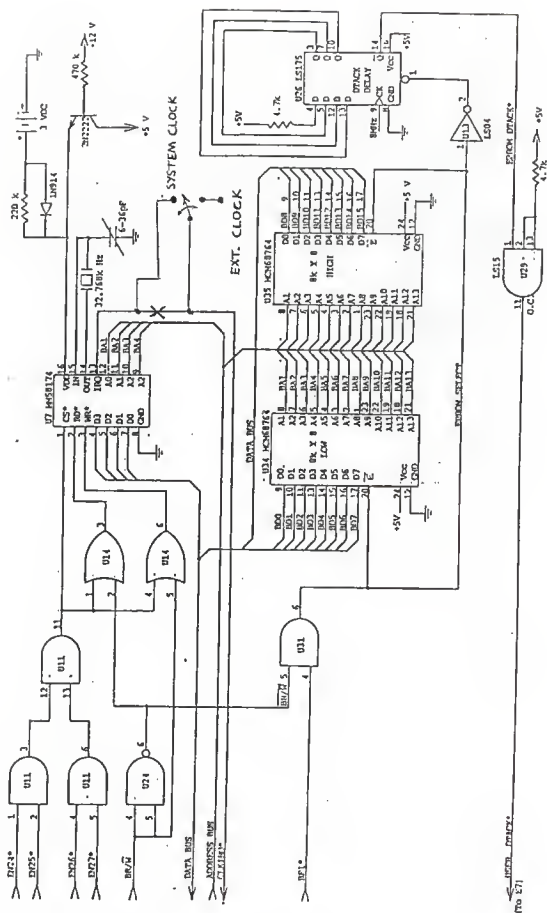
Figure A.4 EEPROM'S AND REAL TIME CLOCK [2]
ADDITION OF EXTERNAL CLOCK SWITCH

110

APPENDIX A (CONTINUED)

CHANGES TO DACI ROUTINES FROM [2] NOTED

.

```
              ORG.        $20900
START         MOVE.L      #8,D3            SEND EIGHT LINE FEEDS
LF1           MOVE.B      #$A,DO           TO THE TERMINAL OUTPUT
              MOVE.B      #248,D7          CHARACTER HANDLER
              TRAP        #14
              DBRA        D3,LF1
              MOVE.B      #$D,DO           SEND CARRIAGE RETURN
              MOVE.B      #248,D7          OUTPUT CHARACTER HANDLER
              TRAP        #14
              MOVE.L      #30,D3           SEND THIRTY SPACES
SP1           MOVE.B      #$20,DO          TO THE TERMINAL
              MOVE.B      #248,D7          OUTPUT CHARACTER HANDLER
              TRAP        #14
              DBRA        D3,SP1
              MOVE.L      #$39D4,A5        DISPLAY "EHSI SYSTEM
              MOVE.L      #$39E8,A6        RUNNING" ON THE TERMINAL
              MOVE.B      #227,D7          OUTPUT STRING HANDLER
              TRAP        #14

PIA#1PA       MOVE.L      #$30001,AO       INIT PIA #1 PORT A OUTPUT
              MOVE.B      #$FF,DO
              JSR         INTPIA
              MOVE.B      #$0F,$30001      INIT CONTROL BITS

PIA#1PB       MOVE.L      #$30005,AO       INIT PIA #1 PORT B
              MOVE.B      #$13,DO      ①   I/P O/P
              JSR         INTPIA
              MOVE.B      #$13,$30005 ②   INIT CONROL BITS

REVISION 3/3/87    ① ENABLES PB4 AS OUTPUT
                   ② INITIALIZE IRQOUT LINE TO HIGH STATE
```

112

```
LOGO        CLR.L       D1              SET UP LOGO FOR DISPLAY
            MOVE.L      #$3A90, AO      ON THE HP DISPLAY
            MOVE.W      (AO)+, D1
            JSR         SNDLST          SEND THE LOGO TO VGD
            MOVE.W      #$C000, DO      SET UP ADDRESS POINTER

SYSON       BTST        #3, $30005      WAIT FOR SYSTEM ON
            BEQ         SYSON           LOW OFF, HIGH ON
            JSR         SNDCMD          SET UP ADDRESS PTR $O
            MOVE.W      #0, DO          SEND A NOP WHERE JUMP
            JSR         SNDCMD          WAS SO LOGO CAN BE
                                        DISPLAYED
SYSUP       MOVE.B      #$65, $30021    SEND SYS UP VEC TO HOST
            JSR         OUTSHAKE_IRQ ①  SEND STROBE TO HOST
            MOVE.B      #$2, D2         SET ERROR FLAG TO ACKERR
            JSR         INSHAKE         WAIT FOR ACKNOWLEDGE
            MOVE.L      #$30021, AO     MAKE PARALLEL PORT INPUT
            MOVE.B      #0, DO
            JSR         INTPIA

ENIRQS      OR.B        #$FO, $30001    ENABLE ALL INTERRUPTS

ENCLKIRQ    MOVE.B      #$OC, $3007F    ENABLE INT INTERVAL

            MOVE.B      #24, $1FAO      PUT # OF ENT IN DATA PAK
MAIN        JSR         ANALOG          PUT ANALOG VAL IN PACK
            NOP                      ② NOP'S ADDED FOF CALL TO
            NOP                         MODIFIED CLOCK_IRQ
            MOVE.B      $30019, $1FB1   PUT BINARY I/P'S IN PACK
            BTST        #3, $30005      CHECK TO SEE IF SYS IS ON
            BNE         MAIN            IF ON, GO THRU MAIN AGAIN

SYSDN       AND.B       #$OF, $30001    DISABLES INTERRUPTS
            MOVE.B      #$2700, SR      DISABLES ALL INT LEVELS
            BSET        #0, $30005      TURN ALARM OFF IF ON

SHUTN       MOVE.L      #$3AA4, AO      DISPLAY SYS SHUTDOWN MSG
            CLR.L       D1
            MOVE.W      (AO)+, D1
            JSR         SNDLST
            MOVE.L      #$30021, AO     MAKE PARALLEL PORT O/P
            MOVE.B      #$FF, DO
            JSR         INTPIA
            MOVE.B      #$66, (AO)      SEND SHUTDN VEC TO HOST
            JSR         OUTSHAKE_IRQ ①  STROBE THE HOST
            MOVE.B      #2, D2          SET ERROR FLAG TO ACKERR
            JSR         INSHAKE         WAIT FOR ACKNOWLEDGE

REVISION 3/3/87         ① CALLS TO OUTSHAKE_IRQ ADDED
                        ② NOP'S ADDED FOR HIGH SPEED OPERATION
```

113

```
              ORG.       $20760
KEYIRQ        MOVEM.L    DO/D2/AO,$9BC    SAVE REGISTERS
              JSR        GETKEY           GET VAL OF KEY PRESSED
              CMPI.B     #37,D2           WAS THERE KEY FOUND
              BEQ        NOKEY            IF NO KEY THEN RETURN
              CMPI.B     #36,D2           CHECK FOR SET CLOCK KEY
              BNE        CONT5            IF NOT SET CLOCK KEY
              JSR        SETCLK           SET THE CLOCK/CALENDAR
              BRA        NOKEY            AFTER CLOCK IS SET, RET
CONT5         MOVE.L     #$30021,AO       CHANGE PARALLEL PORT, O/P
              MOVE.B     #$FF,DO
              JSR        INTPIA
              MOVE.B     D2,$30021        SEND VALID KEY, PAR PORT
              MOVE.B     #2,D2            SET ERROR FLAG TO ACKERR
              JSR        OUTSHAKE_IRQ ①   STROBE THE HOST
              JSR        INSHAKE          WAIT FOR ACKNOWLEDGE
              BSET       #7,$30001        ENABLE PAR PORT TO INT'S
              MOVE.L     #$30021,AO       CHANGE PAR PORT TO I/P
              MOVE.B     #0,DO
              JSR        INTPIA
NOKEY         BCLR       #6,$30001        CLEAR KEYBOARD INTERRUPT
              BSET       #6,$30001
              MOVEM.L    $9BC,DO/D2/AO    RESTORE REGISTERS
              RTS                         RETURN

REVISION 3/3/87      ① CALL TO OUTSHAKE_IRQ ADDED
```

114

```
        MOVE.B      $3006F, DO      READ TENS OF HOURS
        AND.W       #$F, DO         CLEAR DATA4 - DATA7
        MULU.W      #10, DO
        MOVE.B      $3006D, D2      READ THE ONES OF HOURS
        AND.B       #$F, D2         CLEAR DATA4 - DATA7
        ADD.B       D2, DO          ADD TENS AND ONES OF HRS
        MOVE.B      DO, -(AO)       PUT IN THE DATA PACKAGE
        MOVE.B      $30073, DO      READ TENS OF DAYS
        AND.W       #$F, DO         CLEAR DATA4 -  DATA7
        MULU.W      #10, DO
        MOVE.B      $30071, D2      READ THE ONES OF DAYS
        AND.B       #$F, D2         CLEAR DATA4 - DATA7
        ADD.B       D2, DO          ADD TENS AND ONES OF DAYS
        MOVE.B      DO, -(AO)       PUT IN THE DATA PACKAGE
        MOVE.B      $30075, DO      READ DAY OF WEEK
        AND.B       #$F, DO         CLEAR DATA4 -  DATA7
        MOVE.B      DO, -(AO)       PUT IN THE DATA PACKAGE
        MOVE.B      $30079, DO      READ TENS OF MONTHS
        AND.W       #$F, DO         CLEAR DATA4 -  DATA7
        MULU.W      #10, DO
        MOVE.B      $30077, D2      READ THE ONES OF MTHS
        AND.B       #$F, D2         CLEAR DATA4 - DATA7
        ADD.B       D2, DO          ADD TENS AND ONES OF MTHS
        MOVE.B      DO, -(AO)       PUT IN THE DATA PACKAGE
        MOVE.B      $30063, DO      READ DAY OF WEEK
①       MOVE.L      #$30021, AO     MAKE PAR PORT O/P
        MOVE.B      #$FF, DO
        JSR         INTPIA
        MOVE.B      #$60, $30021    SEND THE TIME VEC TO HOST
        JSR         OUTSHAKE_IRQ ②  SEND DATA STROBE
        MOVE.B      #2, D2          SET ERROR FLAG TO ACKERR
        JSR         INSHAKE         WAIT FOR ACKNOWLEDGE
        BSET        #7, $30001      ENABLE PAR PORT FOR INT'S
        MOVE.L      #$30021, AO     CHANGE PAR PORT TO I/P
        MOVE.B      #0, DO
        JSR         INTPIA
FAST    MOVE.B      $3007F, DO      3 READS CLEARS CLOCK IRQ
        MOVE.B      $3007F, DO
        MOVE.B      $3007F, DO
        BCLR        #5, $30001      CLEAR CLOCK INT LATCH
        BSET        #5, $30001
        MOVEM.L     $9BC, DO/D2/AO  RESTORE REGISTERS
        RTS                         RETURN
```

REVISION 3/3/87   ① A JUMP TO FAST CAN BE ADDED HERE FOR
                       HIGH SPEED OPERATION WITH THE CLOCK
                       INTERRUPT DISABLED. CLOCK VALUES
                       WILL BE RETRIEVED AND HOST WILL NOT
                       BE INTERRUPTED.
                     ② CALL TO OUTSHAKE_IRQ ADDED

                            115

APPENDIX B

EHSI USER'S MANUAL

SYSTEM ON PROCEDURE

To bring the EHSI system to its operational state, complete the procedures of the following sections in order.

DACI ON

The procedure to start up the DACI contains the following steps:

STEP 1) Make sure all the ribbon cables are connected to their appropriate ports on the DACI.

STEP 2) Connect the green +5 volt cable to the +5 volt output of the power supply for the Motorola MC68000 Educational Board.

Warning: Never turn off the power supply for the Educational Board or the DACI control program will be lost. This warning will be in affect until the EEPROM's are burned for the DACI control program.

STEP 3) Turn on the power supply for the HP 1345A and

the DACI interface board.

STEP 4) Turn on the Zenith Z-29 video terminal. This
is the terminal connected to the Educational
Board. The power switch is located on the back
of the terminal at the bottom of the right-hand
side.

STEP 5) Press the "CAPS LOCK" key of the Z-29. This
will place the Z-29 in the capital letter mode
which is necessary for entering Educational
Board commands.

STEP 6) Press the black hardware reset button on the
Educational Board.

STEP 7) Press the red software abort button on the
Educational Board.

STEP 8) Make sure the EHSI system switch is in the
"off" position. This switch is located above
the control keyboard and should be in the down
position.

STEP 9) Run the DACI control program by entering the
following command:

GO 2900

Then, hit the RETURN button on the Z-29. The
following message should appear:

EHSI SYSTEM RUNNING

INITIALIZATIONS COMPLETE

117

CONTINUE TO FLIGHT SIMULATOR

If any error messages occur, refer to [2] for
actions to be taken.

This completes the necessary steps to turn on the DACI.
Proceed to the next section.

HOST ON

Execute the following steps to turn on the host side of
the EHSI system:

STEP 1) Turn on the Zenith Z-158 host computer. The
power switch is located on the power strip on
top of the lab bench to the right of Z-158.
This switch will also turn on the Z-158 mon-
itor. The following prompt should be
displayed:

C:\>

STEP 2) Enter the following command at the prompt:
cd\ehsi

Then, hit the **RETURN** button on the Z-158. The
following prompt should be displayed:

C:\ehsi>

STEP 3) Enter the following command at this prompt:
ehsi

Press the **RETURN** button.

118

This completes the necessary steps to turn on host side of the EHSI. Proceed to the next section.

FLIGHT SIMULATOR ON

Execute the following steps to turn on the ATC-610 Flight Simulator:

STEP 1) Get the flight simulator key and insert it into the ignition switch. Turn the key until the switch is in the "on" position.

STEP 2) Push the master switch to the "on" position. this switch is located to the right of the ignition switch.

STEP 3) For instructions on the use of the flight simulator proceed to the ATC-610 Flight Simulator User's Manual.

This completes the necessary steps to turn on the simulator. Proceed to the EHSI ON section.

EHSI ON

Turn the EHSI system switch located above the control keyboard to the "on" position. The switch will be in the up position. The following message will appear on the monitor of the Z-158:

SYSTEM SWITCH ON.

The DATA PAGE will now be displayed on the HP 1345A. The

119

EHSI is now fully operational.   Proceed to the flight simu-
lator.

     This completes the SYSTEM ON PROCEDURE.

SYSTEM OFF PROCEDURE

     To shut down the EHSI system, complete the procedures
of the following sections in order.

DACI OFF

     Turn the EHSI system switch located above the control
keyboard to the "off" position.   The switch will be in the
up position.   The following message will appear on the
monitor of the Z-158 along with the DOS prompt:

          SYSTEM SWITCH OFF.

          C:\ehsi>

A shutdown message will also appear on the HP 1345A.   Pro-
ceed to the next section.

FLIGHT SIMULATOR OFF

     Execute the following steps to turn off the ATC-610
Flight Simulator:

     STEP 1) Push the master switch to the "off" position.
              This switch is located to the right of the ig-
              nition switch.

     STEP 2) Turn the key in the ignition switch to the

"off" position. Remove the key and return it
to the flight simulator log book.

This completes the necessary steps to turn off the
flight simulator. Proceed to the next section.

HOST OFF

Turn off the Zenith Z-158 host computer. The power
switch is located on the power strip on top of the lab bench
to the right of the Z-158. The switch also turns off the Z-
158's monitor. The host should now be in its off state.
Proceed to the next section.

DACI OFF

Execute the following steps to turn off the DACI:

STEP 1) Press the red software abort button on the
Educational Board.

STEP 2) Turn off the Zenith Z-29 video terminal.
This is the terminal connected to the
Educational Board. The power switch is
located on the back of the terminal at the
bottom of the right-hand side.

STEP 3) Turn off the power supply for the HP 1345A and
the DACI interface board.

STEP 4) Disconnect the green +5 volt cable from the
output of the power supply for the Motorola

MC68000 Educational Board.

Warning: Never turn off the power supply for the
        Educational Board or the DACI control program
        will be lost.   This warning will be in affect
        until the EEPROM's are burned for the DACI
        control program.

STEP 5) Disconnect the ribbon cables going to the host,

        flight simulator, and control keyboard.

This completes the SYSTEM OFF PROCEDURE

.

APPENDIX C

EHSI HOST PROGRAM MAINTENANCE

All the host programs for the EHSI are located in the
ehsi directory. To get to the ehsi directory, type in the
following command at the prompt:

C:\>cd\ehsi

Then, press the RETURN key. The following prompt will
appear on the monitor:

C:\ehsi>

The EHSI host programs can now be edited, compiled, and
linked.

EDITING

All editing for the host programs of the EHSI is done
using BRIEF, Version 1.33. The is a full-screen editor
designed specifically for the editing of programs. To
begin editing a file, type the following command at the
prompt:

C:\ehsi>b XXXXXX.c

The program name to be edited should be inserted at XXXXXX.
Press the **RETURN** key to execute the editor. The name of the
file being edited will appear at the top of the screen. To
exit the editor, press the **Alt** key followed by the X key and
answer the questions which appear in the command line at the
bottom of the screen. The various editing commands avail-
able can be seen in Fig. C.1[5] and C.2[5]. This is the
BRIEF Quick Reference Card. For a more detailed description
of the commands refer to the **BRIEF** manual.

COMPILING AND LINKING WITHOUT USING MAKE UTILITY

COMPILING

The Microsoft C compiler, Version 4.0, is the compiler
used for the EHSI. Compiling of the host programs for the
EHSI can be done in two different ways. The first method is
to compile the programs directly. The second method is to
compile the programs from inside the **BRIEF** editor using a
macro.

To compile the programs directly type in the following
command at the prompt:

C:\ehsi>mac XXXXXX;

The program name to be compiled should be inserted at XXXXXX.
Press the **RETURN** key to execute the compilation. The
following message will be displayed:

124

Buffer Commands

Alt-b	Buffer list
Alt-e	Edit file (create buffer)
Alt-f	Display buffer file name
Alt-n	Edit next buffer
Ctrl-minus	Edit previous buffer
Alt-r	Read file into buffer
Alt-o	Change output file name
Alt-w	Write buffer
Alt-minus	Delete current buffer

Search/Translate Commands

Alt-c	Toggle case sensitivity
F5, Alt-s	Search forward
F6, Alt-t	Translate
Shift-F5	Search backward
Shift-F6	Search again
Alt-F5	Incremental search forward
Alt-F6	Toggle regular expressions

Cursor Commands

Down arrow	Down one line
Left arrow	Left one column
Right arrow	Right one column
Up arrow	Up one line
PgDn	Page down
PgUp	Page up
Home	Top of window
End	End of window
Ctrl-PgDn	End of line
Ctrl-PgUp	Beginning of line
Ctrl-Home	Top of buffer
Ctrl-End	End of buffer
Alt-g	Goto line
Ctrl-Left arrow	Previous word
Ctrl-Right arrow	Next word
Ctrl-b	Line to bottom of window
Ctrl-c	Center line in window
Ctrl-d	Scroll down one line
Ctrl-t	Line to top of window
Ctrl-u	Scroll up one line
Shift-PgUp	Left edge of window
Shift-PgDn	Right edge of window

Regular Expressions/Wildcards

!	Zero or more of any characters
?	Any one character
@	Zero or more of previous character or group
{	Begin group
}	End group
\\<num>	Substitute text matched by <num>th group
[	Begin inclusive character group
[~	Begin exclusive character group
]	End character group
-	Define range in character group
		Either the preceding or the following character or group
\\n	Newline character
\\t	Tab character
<, %	Beginning of line
>, $	End of line (excluding newline)
\\c	Place cursor at specified character

Basic Text Commands

Backspace	Backspace and delete/overwrite
Del	Delete character
Enter	Insert new line/Move to next line
Tab	Insert tab/Next tab stop

Figure C.1   BRIEF Quick Reference Card [5]

Alt-d	Delete line
Alt-k	Delete/Kill to end of line
Ctrl-Enter	Open new line after current line

Macro Commands

F7	Record/Stop recording
F8	Play back recording
F9	Load macro file
F10	Execute macro
Shift-F9	Delete macro file

Block Text Commands

Alt-a, Alt-m	Mark
Alt-p	Print
Alt-w	Write to file
Del	Delete
Ins	Paste from scrap
Grey/Keypad minus	Cut to scrap
Gray/Keypad plus	Copy to scrap

Window Commands

F1	Change window
F2	Resize window
F3	Create window
F4	Destroy window
Shift-Down arrow	Change to window below
Shift-Left arrow	Change to left window
Shift-Right arrow	Change to right window
Shift-Up arrow	Change to window above

Miscellaneous Commands

Esc	Escape/Cancel
Alt-h	Help
Alt-i	Toggle insert mode
Alt-u	Undo
Gray/Keypad *	Undo
Alt-v	Display version ID
Alt-x	Exit
Alt-z	Suspend session
Ctrl-Break	Halt search or macro
Ctrl-w	Toggle backup files
Shift-F10	Insert key code in buffer
Alt-F10	Compile program in buffer
Ctrl-n	Locate next error
Ctrl-p	Display previous error text
Ctrl-r	Repeat following command

Figure C.2   BRIEF Quick Reference Card [5]

Microsoft (R) C Compiler Version 4.00
Copyright (C) Microsoft Corp 1984, 1985, 1986. All rights
reserved.

If there are any errors or warnings, the number of each
respectively will also be displayed. The XXXXXX.ERR can
then be viewed to note what error or warning occurred as
well as the location of the error or warning. The error
file can be viewed inside the BRIEF editor or by executing
the type DOS command.

To compile EHSI program inside the BRIEF editor, open
up the file to be compiled as was discussed in the EDITING
section above. Execute the compiler macro by pressing the
Alt key followed by the F10 key. The program will then
begin compiling. The command line at the bottom will show
what compiler is being used and the options that are select-
ed. When compilation is complete, the location of the first
error or warning will be indicated by the cursor. The
command line will display the nature of the error or warn-
ing. To move to the next error or warning, press the Ctrl
key followed by the N key. When all the error and warning
messages have been displayed, the following message will
appear in the command line:

No more errors

To execute the error handler again, repeat the key sequence
for looking at the next error or warning.

Both of the key sequences for compiling and viewing the

127

errors and warnings are under the Miscellaneous Commands of
the quick reference card in Fig. C.1[5].

WARNING

When the Make Utility is not being used to compile and
edit, special care will have to be taken when making modifi-
cations to the clock_pkg and alarm_pkg data structures of
the host EHSI program.   These structures are in a file
entitled data_str.h which is included in the following rou-
tines:

    1) insert_new_freq

    2) set_timer

    3) dat_pg_dynamic

    4) nav_pg_dynamic

    5) ils_pg_dynamic

    6) altitude

    7) dme

    8) airspeed

    9) timer

   10) climb_rate

   11) set_estimated_wind

   12) set_altitude

   13) ehsi

These routines will have to be recompiled and added to the
library whenever a change is made to these data structures.

If this procedure is not followed, the results of the host EHSI program will be unpredictable.

LIBRARY

All the routines used by the host EHSI program are contained in a library which is used when linking the EHSI routine. Linking the EHSI routine will be discussed in the next section. The libraries for the host EHSI program are called **ehsi.lib** and **key_ehsi.lib**. When a new compiled routine is ready to be added to the library, type in the following command at the prompt:

C:\ehsi>lib ehsi+XXXXXX

Insert the name of the routine to be added at XXXXXX. Press the **RETURN** key to begin installation of the routine into the library. The following will then appear:

Microsoft (R) Library Manager Version 3.04
Copyright (C) Microsoft Corp 1983,1984,1985,1986 All rights reserved.

List File:

Enter **cross** after the **List file:** prompt and hit the **RETURN** key. This will provide a cross listing of all the files presently contained in the library.

To add a new version of a routine which already exists in the library, type in the following at the prompt:

C:\ehsi>lib ehsi-XXXXXX+YYYYYY

Insert the name of the old version of the routine contained

129

in the library at XXXXXX.   Insert the name of the new ver-
sion of the routine at YYYYYY.   Usually,   XXXXXX and YYYYYY
will be the same.   Then, press the **RETURN** key and proceed as
before.

LINKING

To   produce the executable file for the host EHSI   pro-
gram,   the EHSI routine must be linked to the other routines
of   the EHSI.   This is done by typing in the following co-
mmand at the prompt:

            C:\ehsi>link ehsi XXXXXX/stack:4000

Then,   press the **RETURN** key.   Insert any new routine being
tested at XXXXXX.   Note, more than one routine can be linked
with the EHSI routine.   Put a space between any additional
routines being tested.   The stack option is used to increase
the   stack   size allotted for the   program.   The   following
message will now appear:

Microsoft (R) Overlay Linker Version 3.51
Copyright (C) Microsoft Corp 1983,1984,1985,1986 All rights
reserved.

Run File [EHSI.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:

Press the **RETURN** key after the Run File and List File
prompts.   For an explanation of these,   refer to the linker
portion of the manuals.   Enter ehsi and key_ehsi after the
Libraries prompt and press the RETURN key.   The new routines

130

being tested and the routines in **ehsi.lib** and **key_ehsi.lib** will now be linked to the EHSI routine. This will produce an executable file. Any error messages will be displayed as the linking process progresses.

COMPILING AND LINKING WITH THE MAKE UTITILITY

Compiling and linking of the EHSI host programs can be done in one step with the make utility. The make file for the EHSI can be seen at the end of the appendix and is entitled **ehsi** in the **ehsi** directory. This file contains the directory names of the EHSI host programs and the dependencies for each of the programs. The file also contains the the link list for the EHSI program and other necessary parameters.

The make utility recompiles any routine whose source file is newer than the latest object file for the routine. Routines which are also dependent on certain include files will also be recompiled when the include file is newer than the latest object file for the routine. After compiling all the routines specified in the make file, the EHSI program is linked with the link list specified in the make file along with the listed parameters.

To execute the make utility available, enter the following command at the prompt:

C:\ehsi>**make ehsi**

Press the RETURN key to execute the compilation and linking of EHSI programs into an exectuable EHSI file. If no errors occur while the make utility progresses through the compilation and linking process, a new executable file will be produced in the directory entitled ehsi.exe. If there are any errors or warnings, the number of each respectively will be displayed. The XXXXXX.ERR can then be viewed to note what error or warning occurred as well as the location of the error or warning. The error file can be viewed insided the BRIEF editor or by executing the type DOS command.

Note, when any new routines are added to the EHSI program, changes must be made to the ehsi make file. Refer to the programming user's manual for formats when making these changes to the make file.

Warning, when compiling programs which are added to the ehsi.lib and key_ehsi.lib without using the make utility, make sure to erase the latest object file generated for the source file being compiled. If this is not done, the latest version will not be added to the appropriate library when the make utility is run and the latest version will not be linked with the EHSI program.

APPENDIX C (CONT.)

EHSI MAKE FILE

File: ehsi

```
/**************************************************************
 *
 *   SOURCE FILE:      ehsi
 *
 *
 *   FUNCTION:         None.
 *
 *
 *   DESCRIPTION:      Make file for the EHSI host program.
 *                     Used when running make utility.
 *
 *
 *
 *   DOCUMENTATION
 *   FILES:            None.
 *
 *
 *   ARGUMENTS:        None.
 *
 *
 *   RETURN:           None.
 *
 *
 *   FUNCTIONS
 *   CALLED:           None.
 *
 *
 *   AUTHOR:           DAVE GRUENBACHER AND CHUCK ROBERTSON
 *
 *
 *   DATE CREATED:     10Apr87      Version 1.0
 *
 *
 *   REVISIONS:        None.
 *
 *
 **************************************************************/

insert.obj   : insert.c
        msc insert;
        lib ehsi-+insert,ehsi.crs;

time_gen.obj : time_gen.c
        msc time_gen;
        lib ehsi-+time_gen,ehsi.crs;

line.obj     : line.c
        msc line;
        lib ehsi-+line,ehsi.crs;
```

134

```
clim_box.obj : clim_box.c
        msc clim_box;
        lib ehsi-+clim_box, ehsi.crs;

clim_hsh.obj : clim_hsh.c
        msc clim_hsh;
        lib ehsi-+clim_hsh, ehsi.crs;

climrate.obj : climrate.c
        msc climrate;
        lib ehsi-+climrate, ehsi.crs;

climfilt.obj : climfilt.c climfilt.h
        msc climfilt;
        lib ehsi-+climfilt, ehsi.crs;

arc_circ.obj : arc_circ.c
        msc arc_circ;
        lib ehsi-+arc_circ, ehsi.crs;

str_gen.obj  : str_gen.c
        msc str_gen;
        lib ehsi-+str_gen, ehsi.crs;

plane.obj    : plane.c
        msc plane;
        lib ehsi-+plane, ehsi.crs;

waypoint.obj : waypoint.c
        msc waypoint;
        lib ehsi-+waypoint, ehsi.crs;

vortac.obj   : vortac.c
        msc vortac;
        lib ehsi-+vortac, ehsi.crs;

box.obj      : box.c
        msc box;
        lib ehsi-+box, ehsi.crs;

compass.obj  : compass.c
        msc compass;
        lib ehsi-+compass, ehsi.crs;

clim_aro.obj : clim_aro.c
        msc clim_aro;
        lib ehsi-+clim_aro, ehsi.crs;

arrow.obj    : arrow.c
```

```
        msc arrow;
        lib ehsi-+arrow, ehsi.crs;

ndb.obj        : ndb.c
        msc ndb;
        lib ehsi-+ndb, ehsi.crs;

heading.obj  : heading.c
        msc heading;
        lib ehsi-+heading, ehsi.crs;

runway.obj    : runway.c
        msc runway;
        lib ehsi-+runway, ehsi.crs;

zero_pad.obj : zero_pad.c
        msc zero_pad;
        lib ehsi-+zero_pad, ehsi.crs;

timer.obj     : timer.c data_str.h
        msc timer;
        lib ehsi-+timer, ehsi.crs;

altitude.obj : altitude.c data_str.h
        msc altitude;
        lib ehsi-+altitude, ehsi.crs;

dme.obj        : dme.c data_str.h
        msc dme;
        lib ehsi-+dme, ehsi.crs;

ndb_angl.obj : ndb_angl.c
        msc ndb_angl;
        lib ehsi-+ndb_angl, ehsi.crs;

airspeed.obj : airspeed.c data_str.h
        msc airspeed;
        lib ehsi-+airspeed, ehsi.crs;

climrate.obj : climrate.c  data_str.h
        msc climrate;
        lib ehsi-+climrate, ehsi.crs;

ils_cmps.obj : ils_cmps.c
        msc ils_cmps;
        lib ehsi-+ils_cmps, ehsi.crs;

hdg_brg.obj  : hdg_brg.c
        msc hdg_brg;
        lib ehsi-+hdg_brg, ehsi.crs;
```

```
key_alrm.obj : key_alrm.c
        msc key_alrm;
        lib key_ehsi-+key_alrm, key_ehsi.crs;

key_dat.obj  : key_dat.c
        msc key_dat;
        lib key_ehsi-+key_dat, key_ehsi.crs;

key_nav.obj  : key_nav.c
        msc key_nav;
        lib key_ehsi-+key_nav, key_ehsi.crs;

key_ils.obj  : key_ils.c
        msc key_ils;
        lib key_ehsi-+key_ils, key_ehsi.crs;

key_entr.obj : key_entr.c
        msc key_entr;
        lib key_ehsi-+key_entr, key_ehsi.crs;

key_cler.obj : key_cler.c
        msc key_cler;
        lib key_ehsi-+key_cler, key_ehsi.crs;

key_freq.obj : key_freq.c data_str.h
        msc key_freq;
        lib key_ehsi-+key_freq, key_ehsi.crs;

key_stmr.obj : key_stmr.c data_str.h
        msc key_stmr;
        lib key_ehsi-+key_stmr, key_ehsi.crs;

key_math.obj : key_math.c
        msc key_math;
        lib key_ehsi-+key_math, key_ehsi.crs;

key_buff.obj : key_buff.c
        msc key_buff;
        lib key_ehsi-+key_buff, key_ehsi.crs;

key_alt.obj  : key_alt.c data_str.h
        msc key_alt;
        lib key_ehsi-+key_alt, key_ehsi.crs;

key_wind.obj : key_wind.c data_str.h
        msc key_wind;
        lib key_ehsi-+key_wind, key_ehsi.crs;

key_cmd3.obj : key_cmd3.c
```

```
        msc key_cmd3;
        lib key_ehsi-+key_cmd3,key_ehsi.crs;

dat_pg_s.obj   : dat_pg_s.c datpg_xy.h
        msc dat_pg_s;

dat_pg_d.obj    : dat_pg_d.c data_str.h datpg_xy.h
        msc dat_pg_d;

nav_pg_s.obj   : nav_pg_s.c navpg_xy.h
        msc nav_pg_s;

nav_pg_d.obj   : nav_pg_d.c data_str.h navpg_xy.h
        msc nav_pg_d;

ils_pg_s.obj   : ils_pg_s.c ilspg_xy.h
        msc ils_pg_s;

ils_pg_d.obj   : ils_pg_d.c data_str.h ilspg_xy.h
        msc ils_pg_d;

int_ehsi.obj : int_ehsi.asm
        masm int_ehsi;

com_ehsi.obj : com_ehsi.asm
        masm com_ehsi;

ehsi.obj      : ehsi.c data_str.h
        msc ehsi;


ehsi.exe : ehsi.obj dat_pg_s.obj nav_pg_s.obj ils_pg_s.obj\
           dat_pg_d.obj nav_pg_d.obj nav_pg_d.obj\
           com_ehsi.obj int_ehsi.obj ehsi.lib key_ehsi.lib
link ehsi dat_pg_s nav_pg_s ils_pg_s dat_pg_d nav_pg_d
          ils_pg_d com_ehsi int_ehsi/stack:4000,ehsi,ehsi,
          ehsi.lib key_ehsi.lib;
```

APPENDIX D

EHSI HOST PROGRAM SOFTWARE LISTINGS

```
/***************************************************************
 *
 *  SOURCE FILE:     ehsi.c
 *
 *
 *  FUNCTION:        main()
 *
 *
 *  DESCRIPTION:     Controls the actions taken on receipt
 *                   of an interrupt vector from the
 *                   interrupt vector stack. Iniatializa-
 *                   tion and restoration are also perform-
 *                   ed from this routine. The data pack-
 *                   age, SCREEN array interrupt vector
 *                   stack, and calculator stack are
 *                   declared within this routine.
 *
 *                    .
 *  DOCUMENTATION
 *  FILES:           None.
 *
 *
 *  ARGUMENTS:       None.
 *
 *
 *  RETURN:          None.
 *
 *
 *  FUNCTIONS
 *  CALLED:          ' None.
 *
 *
 *  AUTHOR:          Dave Gruenbacher & Chuck Robertson
 *
 *
 *  DATE CREATED:    19Jan87      Version 1.0
 *
 *
 *  REVISIONS:       None.
 *
 *
 ***************************************************************/
#define  ehsi_main
#include <stdio.h>
#include "data_str.h"

void main()
{
    static unsigned short int_depth = 0, int_stack[90]={0};
    int    page_number = 1, i, int_number;
```

140

```
char    key_buffer[50];
double  x_buffer, y_buffer;

void    INITIALIZE();
void    RESTORE();
void    dat_pg_dynamic();
void    nav_pg_dynamic();
void    ils_pg_dynamic();
void    dat_pg_static();
void    nav_pg_static();
void    ils_pg_static();
void    update_key_buffer();
void    roll_stack();
void    set_altitude();
void    set_estimated_wind();
void    exit();
void    clear_stack();
void    insert_new_freq();
void    set_timer();
void    reset_alarm();
void    do_math();
void    call_cmd3();
void    display_ils_page();
void    display_data_page();
void    display_nav_page();

CLOCK_PKG clock_pkg;
ALARM_PKG alarm_pkg;

clock_pkg.timer_min = 0;
clock_pkg.timer_sec = 0;
clock_pkg.time_out_min = 0;
clock_pkg.time_out_sec = 0;
clock_pkg.adf_freq = 242.0;
clock_pkg.com1_freq = 119.1;
clock_pkg.com2_freq = 121.9;
clock_pkg.vor1_freq = 112.6;
clock_pkg.vor2_freq = 110.1;
clock_pkg.assigned_altitude = 0;
clock_pkg.mda_dh = 0;
clock_pkg.estimated_wind = 0;
clock_pkg.timer_operation_flag = NULL_TIMER;
clock_pkg.timer_status_flag = TIMER_OFF;
clock_pkg.math_operation_flag = 0;

alarm_pkg.airspeed_alarm_flag = ALARM_OFF;
alarm_pkg.assigned_altitude_alarm_flag = ALARM_OFF;
alarm_pkg.mda_dh_alarm_flag = ALARM_OFF;
alarm_pkg.time_out_alarm_flag = ALARM_OFF;
alarm_pkg.alarm_status_flag = ALARM_OFF;
```

141

```c
    alarm_pkg.assigned_altitude_enable_flag = DISABLED;
    alarm_pkg.mda_dh_enable_flag = DISABLED;

    INITIALIZE(&int_depth, int_stack);
    key_buffer[0] = '\0';

    for (;;)
        {
        if (int_depth != 0)
            {
/*          if (int_stack[0] != 0x60)          */
                printf("%X %X\n", int_depth, int_stack[0]);
            int_number = int_stack[0];
            int_depth -= 1;
            for (i=0;i!=int_depth;i++)
                int_stack[i] = int_stack[i+1];

        switch (int_number) {

        case 0x60:
            int_number = 0;

            switch (page_number) {

                case 1:
                    dat_pg_dynamic(&clock_pkg, &alarm_pkg);
                    break;

                case 2:
                    nav_pg_dynamic(&clock_pkg, &alarm_pkg);
                    break;

                case 3:
                    ils_pg_dynamic(&clock_pkg, &alarm_pkg);
                    break;

                default:
                    break;
            }
            break;

        case 0x65:
            int_number = 0;
            printf("\n\nSYSTEM SWITCH ON.\n\n");
            /* set up vector table */
            dat_pg_static();    /* Install data page static
                                   mem. */
            for (i=0;i!=100;i++);
            nav_pg_static();    /* Install nav. page static
                                   mem. */
```

```
    for (i=0;i!=100;i++);
    ils_pg_static();    /* Install ils page static
                            mem.   */
    break;

case 0x66:
    RESTORE();
    printf("\n\nSYSTEM SWITCH OFF.\n\n");
    exit();
    break;


default:
    if ((int_number == 0) || (int_number > 0x23))
        break;

    switch (int_number) {

    case 0x04:              /* 0 received from keypad */
    case 0x05:              /* . received from keypad */
    case 0x0A:              /* 1 received from keypad */
    case 0x0B:              /* 2 received from keypad */
    case 0x0C:              /* 3 received from keypad */
    case 0x10:              /* 4 received from keypad */
    case 0x11:              /* 5 received from keypad */
    case 0x12:              /* 6 received from keypad */
    case 0x16:              /* 7 received from keypad */
    case 0x17:              /* 8 received from keypad */
    case 0x18:              /* 9 received from keypad */
        /* clear the buffer if a math operation was
            just completed.                      */
        if (clock_pkg.math_operation_flag == 1)
            {
            key_buffer[0] = '\0';
            clock_pkg.math_operation_flag = 0;
            }

        update_key_buffer(int_number, key_buffer);
        int_number = 0;
        break;

    case 0x02:              /* ENTER hit on keypad    */
        roll_stack(key_buffer,&x_buffer,&y_buffer);
        int_number = 0;
        break;

    case 0x08:              /* CLEAR hit on keypad    */
        int_number = 0;
        clear_stack(key_buffer,&x_buffer,&y_buffer);
        break;
```

143

```
            case 0x01:              /* new COM1 freq. entered */
            case 0x07:              /* new COM2 freq. entered */
            case 0x0D:              /* new VOR1 freq. entered */
            case 0x13:              /* new VOR2 freq. entered */
            case 0x19:              /* new ADF  freq. entered */
                insert_new_freq(int_number,key_buffer,
                                &clock_pkg);
                int_number = 0;
                break;

            case 0x1B:              /* new mda/dh entered     */
            case 0x1C:              /* new asgn. alt. entered */
                set_altitude(int_number,key_buffer,
                             &clock_pkg,&alarm_pkg);
                int_number = 0;
                break;

            case 0x1D:              /* new est. wind entered */
                int_number = 0;
break;              set_estimated_wind(key_buffer,&clock_pkg);

            case 0x22:              /* SET TIMER hit          */
                int_number = 0;
                set_timer(key_buffer,&clock_pkg);
                clock_pkg.timer_operation_flag = SET_TIMER;
                break;

            case 0x1F:              /* START TIMER hit        */
                int_number = 0;
                clock_pkg.timer_operation_flag
                                = START_TIMER;
                break;

            case 0x23:              /* RESET TIMER hit        */
                int_number = 0;
                clock_pkg.timer_operation_flag
                                = RESET_TIMER;
                break;

            case 0x1E:              /* SET/RST ALRM hit       */
                int_number = 0;
                reset_alarm();
                break;

            case 0x03:              /* div  key hit on keypad */
            case 0x09:              /* mult key hit on keypad */
            case 0x0F:              /* add  key hit on keypad */
            case 0x15:              /* sub  key hit on keypad */
                do_math(int_number,key_buffer,&x_buffer,
                        &y_buffer);
```

```
                int_number = 0;
                clock_pkg.math_operation_flag = 1;
                break;

            case 0x21:
                call_cmd3(key_buffer, &x_buffer, &y_buffer);
                int_number = 0;
                break;

            case 0x1A:
                int_number = 0;
                page_number = 1;
                display_data_page();
                break;

            case 0x14:
                int_number = 0;
                page_number = 2;
                display_nav_page();
                break;

            case 0x0E:
                int_number = 0;
                page_number = 3;
                display_ils_page();
                break;

            default:
                int_number = 0;
                break;
            }
            break;
        }
    }
}
}/* end main */
```

```
/*****************************************************************
 *
 *   SOURCE FILE:      dat_pg.c
 *
 *
 *   FUNCTION:         dat_pg_static()
 *
 *
 *   DESCRIPTION:      This routine generates the static
 *                     data for the DATA PAGE.  It is only
 *                     called once when the SYSTEM SWITCH is
 *                     turned on.
 *
 *
 *   DOCUMENTATION
 *   FILES:            None.
 *
 *
 *   ARGUMENTS:        None.
 *
 *
 *   RETURN:           None.
 *
 *
 *   FUNCTIONS
 *   CALLED:           UPDATE_SCREEN()
 *                     strcpy()
 *                     string_gen()
 *                     insert()
 *                     box()
 *                     line()
 *                     arc_circ()
 *                     climb_box()
 *                     clim_hsh()
 *
 *
 *   AUTHOR:           CHUCK ROBERTSON
 *
 *
 *   DATE CREATED:     15Jan87      Version 1.0
 *
 *
 *   REVISIONS:        None.
 *
 *
 *****************************************************************/
#include<stdlib.h>
#include<string.h>
#include<stdio.h>
#include "datpg_xy.h"
```

146

```c
#include "data_str.h"

void dat_pg_static()
{
    unsigned short conv[300];       /* conversion array */
    unsigned short address_ptr;     /* address pointer */
    unsigned short line_char;       /* line characteristic */
    unsigned short internal_jmp;    /* HP internal jump */
    int XO, YO;                     /* plot coordinates */
    int p;                          /* conversion array
                                                    index */
    int length;                     /* insertion length */
    int i;                          /* loop counter */
    int total_sent;                 /* total words sent to
                                        vector memory      */
    int error;
    char final_string[30];          /* string arrays */
    char cat_string[10];
    double size;                    /* character size */

    int  insert();
    void string_gen();
    void box();
    void line();
    void arc_circ();
    void climb_box();
    void clim_hsh();

    address_ptr = 0xC000;       /* point at first byte in
                                    vector memory              */
    internal_jmp = 0x8001;      /* store jump to 0001 in
                                    vector memory              */
    line_char = 0x7818;         /* set line characteristics */
    size = 1.5;                 /* set character size */

    p=0;
    SCREEN[p++] = address_ptr;
    SCREEN[p++] = internal_jmp;
    SCREEN[p++] = line_char;

        strcpy(final_string, "HEADING:");
        string_gen(final_string, heading_XO, heading_YO,
                    size, &length, conv);
        p = insert(p, length, conv);

        strcpy(final_string, "AIRSPEED:      CAS");
        string_gen(final_string, airspeed_CAS_XO,
                    airspeed_CAS_YO, size, &length, conv);
        p = insert(p, length, conv);
```

147

```
box(col1, 1890, size, length-2, &length, conv);
p = insert(p, length, conv);

strcpy(final_string, "                    TAS");
string_gen(final_string, airspeed_TAS_XO,
           airspeed_TAS_YO, size, &length, conv);
p = insert(p, length, conv);

strcpy(final_string, "GNDSPEED:");
string_gen(final_string, gndspeed_XO,
           gndspeed_YO, size, &length, conv);
p = insert(p, length, conv);

strcpy(final_string, "ASSIGNED:");
string_gen(final_string, assigned_XO,
           assigned_YO, size, &length, conv);
p = insert(p, length, conv);

strcpy(final_string, "ALTITUDE:    ");
string_gen(final_string, altitude_XO,
           altitude_YO, size, &length, conv);
p = insert(p, length, conv);

box(altitude_XO, altitude_YO, size, length-2,
    &length, conv);
p = insert(p, length, conv);

strcpy(final_string, "MDA/DH:");
string_gen(final_string, mda_dh_XO,
           mda_dh_YO, size, &length, conv);
p = insert(p, length, conv);

strcpy(final_string, "TIMER:    ");
string_gen(final_string, timer_XO,
           timer_YO, size, &length, conv);
p = insert(p, length, conv);

box(timer_XO, timer_YO, size, length-2, &length, conv);
p = insert(p, length, conv);

strcpy(final_string, "TIME-OUT: ");
string_gen(final_string, time_out_XO,
           time_out_YO, size, &length, conv);
p = insert(p, length, conv);

strcpy(final_string, "COM1:    ");
string_gen(final_string, com1_XO, com1_YO,
           size, &length, conv);
p = insert(p, length, conv);
```

```c
box(com1_XO, com1_YO, size, length-2, &length, conv);
p = insert(p, length, conv);

strcpy(final_string, "COM2:");
string_gen(final_string, com2_XO, com2_YO,
           size, &length, conv);
p = insert(p, length, conv);

strcpy(final_string, "NAV1:");
string_gen(final_string, nav1_XO,
           nav1_YO, size, &length, conv);
p = insert(p, length, conv);

strcpy(final_string, "NAV2:");
string_gen(final_string, nav2_XO,
           nav2_YO, size, &length, conv);
p = insert(p, length, conv);

strcpy(final_string, "RNAV:WP1   . /    .");
string_gen(final_string, rnav_wp1_XO,
           rnav_wp1_YO, size, &length, conv);
p = insert(p, length, conv);

strcpy(final_string, "     WP2   . /    .");
string_gen(final_string, rnav_wp2_XO,
           rnav_wp2_YO, size, &length, conv);
p = insert(p, length, conv);

strcpy(final_string, "ADF:");
string_gen(final_string, adf_XO, adf_YO, size,
           &length, conv);
p = insert(p, length, conv);

strcpy(final_string, "TEMP:");
string_gen(final_string, temp_XO,
           temp_YO, size, &length, conv);
p = insert(p, length, conv);

strcpy(final_string, "BAROMETER:");
string_gen(final_string, barometer_XO,
           barometer_YO, size, &length, conv);
p = insert(p, length, conv);

/* PUT UP REAL TIME CLOCK INFORMATION */

strcpy(final_string, "TIME:");
string_gen(final_string, time_XO,
           time_YO, size, &length, conv);
p = insert(p, length, conv);
```

149

```c
strcpy(final_string, "EDT");
string_gen(final_string, time_edt_XO,
           time_edt_YO, size, &length, conv);
p = insert(p, length, conv);

strcpy(final_string, "ZULU");
string_gen(final_string, time_zulu_XO,
           time_zulu_YO, size, &length, conv);
p = insert(p, length, conv);

strcpy(final_string, "SINCE L/O");
string_gen(final_string, time_since_lo_XO,
           time_since_lo_YO, size, &length, conv);
p = insert(p, length, conv);

/* PUT UP STATIC INFORMATION FOR CLIMB
   INDICATOR                            */

/* climb indicator lettering */

strcpy(final_string, "C");
string_gen(final_string, C_XO, C_YO, size, &length,
           conv);
p = insert(p, length, conv);

strcpy(final_string, "L");
string_gen(final_string, L_XO, L_YO, size, &length,
           conv);
p = insert(p, length, conv);

strcpy(final_string, "I");
string_gen(final_string, I_XO, I_YO, size, &length,
           conv);
p = insert(p, length, conv);

strcpy(final_string, "M");
string_gen(final_string, M_XO, M_YO, size, &length,
           conv);
p = insert(p, length, conv);

strcpy(final_string, "B");
string_gen(final_string, B_XO, B_YO, size, &length,
           conv);
p = insert(p, length, conv);

/* climb indicator arc */

arc_circ(climb_arc_XO, climb_arc_YO, -20.0, 20.0,
         1000, &length, conv);
p = insert(p, length, conv);
```

```
            /* climb indicator center lines */

            line(climb_centerline1_X0,climb_centerline1_Y0,
                 climb_centerline1_X1,climb_centerline1_Y1,
                 &length,conv);
            p = insert(p,length,conv);

            line(climb_centerline2_X0,climb_centerline2_Y0,
                 climb_centerline2_X1,climb_centerline2_Y1,
                 &length,conv);
            p = insert(p,length,conv);

            line(climb_centerline3_X0,climb_centerline3_Y0,
                 climb_centerline3_X1,climb_centerline3_Y1,
                 &length,conv);
            p = insert(p,length,conv);

            line(climb_centerline4_X0,climb_centerline4_Y0,
                 climb_centerline4_X1,climb_centerline4_Y1,
                 &length,conv);
            p = insert(p,length,conv);

            line(climb_centerline5_X0,climb_centerline5_Y0,
                 climb_centerline5_X1,climb_centerline5_Y1,
                 &length,conv);
            p = insert(p,length,conv);

            /* box climb indicator lettering */

            climb_box(climb_arc_X0,climb_arc_Y0,&length,conv);
            p = insert(p,length,conv);

            /* climb indicator arc hash marks */

            clim_hsh(climb_arc_X0,climb_arc_Y0,-20.0,20.0,
                     1000,&length,conv);
            p = insert(p,length,conv);

            /* COMMAND LINE */

            line(command_line_X0,command_line_Y0,
                 command_line_X1,command_line_Y1,
                 &length,conv);
            p = insert(p,length,conv);

SCREEN[p++] = 0x8700;
SCREEN[p++] = 0xC700;
SCREEN[p++] = 0x0000;
SCREEN[p++] = 0x8FFF;
```

151

```
SCREEN[p++] = 0xCE00;
SCREEN[p++] = 0x0000;
SCREEN[p++] = 0x8FFF;
SCREEN[p++] = 0xFFFF;

error = 1;
while (error != 0)
    error = SEND_SCREEN( );

}    /* end of dat_pg_static */
```

```
/****************************************************************
 *
 *   SOURCE FILE:        dat_pg_d.c
 *
 *
 *   FUNCTION:           dat_pg_dynamic()
 *
 *
 *   DESCRIPTION:        This function generates all the
 *                       dynamic data for the DATA PAGE. It
 *                       retrieves raw data from the data
 *                       package and makes necessary conver-
 *                       sions. Presently it is displaying
 *                       the following:
 *
 *                       1) real time clock
 *                       2) plane heading
 *                       3) present altitude
 *                       4) rate of climb and climb arrow
 *                       5) airspeed
 *                       6) adf frequency
 *                       7) com frequencies
 *                       8) nav frequencies
 *                       9) timer and time_out functions
 *                       10) alarm conditions
 *
 *
 *   DOCUMENTATION
 *   FILES:              None.
 *
 *
 *   ARGUMENTS:          clock_pkg : pointer to clock package
 *                       alarm_pkg : pointer to alarm package
 *
 *
 *   RETURN:             None.
 *
 *
 *   FUNCTIONS
 *   CALLED:             GET_DATA_PACKAGE()
 *                       TOGGLE_ALARM_SWITCH()
 *                       UPDATE_SCREEN()
 *                       insert()
 *                       string_gen()
 *                       time_string_gen()
 *                       get_heading()
 *                       clim_arrow()
 *                       box()
 *                       zero_pad()
 *                       timer()
```

```
    *                       airspeed()
    *                       altitude()
    *                       climb_rate()
    *
    *
    *   AUTHOR:             CHUCK ROBERTSON
    *
    *
    *   DATE CREATED:       01Feb87     Version 1.0
    *
    *
    *   REVISIONS:          None.
    *
    *
    ***********************************************************/
#include "data_str.h"
#include "datpg_xy.h"
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<stdio.h>

void dat_pg_dynamic(clock_pkg, alarm_pkg)

CLOCK_PKG *clock_pkg;
ALARM_PKG *alarm_pkg;

{
    unsigned short conv[300];        /* conversion array */
    int p;                           /* conversion array
                                        index */
    int length;                      /* insertion length */
    int i;                           /* loop counter */
    int vert_speed;                  /* rate of climb */
    int plane_heading_deg;           /* plane heading */
    int times;                       /* request counter */
    int error;                       /* masm return error
                                        code */
    int page_number = 1;             /* page being
                                        displayed */

    char final_string[20];           /* conversion string
                                        arrays */
    char cat_string[5];

    double size;                     /* character size */
    double climb_degrees;            /* angle of climb
                                        arrow */

    int  insert();
```

154

```c
        void string_gen();
        void time_string_gen();
        void get_heading();
        void clim_arrow();
        void box();
        void zero_pad();
        void timer();
        void airspeed();
        void altitude();
        void climb_rate();

        times = 0;
        error = 1;
        while (error != 0)
            {
            times++;
            error = GET_DATA_PACKAGE( );
            }
/*      if(times != 1)
          printf("CK DP times = %d\n",times);        */

        p=0;
        SCREEN[p++] = 0xC700;
        SCREEN[p++] = 0x7818;                    /* set line size */

        size = 1.5;

        /* CONVERT TIME TO CHAR STRING */

        time_string_gen(final_string);

        string_gen(final_string, time_edt_X1, time_edt_Y1, size,
                   &length, conv);
        p = insert(p, length, conv);

        /* PUT UP HEADING */

        get_heading(&plane_heading_deg);

        zero_pad((double)(plane_heading_deg), 100.0, 6,
                 "int", final_string);
        string_gen(final_string, heading_X1, heading_Y1, size,
                   &length, conv);
        p = insert(p, length, conv);

        /* PUT UP ASSIGNED ALTITUDE */

        zero_pad((double)(clock_pkg->assigned_altitude), 1000.0,
                 6, "int", final_string);
```

155

```c
if(alarm_pkg->assigned_altitude_enable_flag == DISABLED)
   string_gen(final_string, assigned_X1, assigned_Y1,
               size, &length, conv);
else
   {
   strcat(final_string, " ALM");
   string_gen(final_string, assigned_X1, assigned_Y1,
               size, &length, conv);
   }

p = insert(p, length, conv);

/*   PUT UP ALTITUDE */

altitude(clock_pkg, alarm_pkg, altitude_X1, altitude_Y1,
         &length, conv);
p = insert(p, length, conv);

/*   PUT UP MDA/DH */

zero_pad((double)(clock_pkg->mda_dh), 1000.0, 6, "int",
         final_string);

if(alarm_pkg->mda_dh_enable_flag == DISABLED)
   string_gen(final_string, mda_dh_X1, mda_dh_Y1, size,
               &length, conv);
else
   {
   strcat(final_string, "   ALM");
   string_gen(final_string, mda_dh_X1, mda_dh_Y1, size,
               &length, conv);
   }

p = insert(p, length, conv);

/*   PUT UP CLIMB VALUES   */

climb_rate(clock_pkg, alarm_pkg, vert_speed_X0,
           vert_speed_Y0, &vert_speed,
           &length, conv);
p = insert(p, length, conv);

box(vert_speed_X0, vert_speed_Y0, size, length-2,
    &length, conv);
p = insert(p, length, conv);

/* calculate climb arrow */
climb_degrees = (double)(vert_speed / 100.0);
clim_arrow(climb_arc_X0, climb_arc_Y0, climb_degrees,
           1000, &length, conv);
```

```c
p = insert(p, length, conv);

/*   PUT UP AIRSPEED   */

airspeed(page_number, clock_pkg, alarm_pkg,
         airspeed_CAS_X1, airspeed_CAS_Y1,
         &length, conv);
p = insert(p, length, conv);

/*   PUT UP ADF   */

zero_pad(clock_pkg->adf_freq, 100.0, 6, "int",
         final_string);
string_gen(final_string, adf_X1, adf_Y1, size, &length,
           conv);
p = insert(p, length, conv);

/*   PUT UP COM1   */

zero_pad(clock_pkg->com1_freq, 100.0, 6, "double",
         final_string);
string_gen(final_string, com1_X1, com1_Y1, size, &length,
           conv);
p = insert(p, length, conv);

/*   PUT UP COM2   */

zero_pad(clock_pkg->com2_freq, 100.0, 6, "double",
         final_string);
string_gen(final_string, com2_X1, com2_Y1, size, &length,
           conv);
p = insert(p, length, conv);

/*   PUT UP VOR1   */

zero_pad(clock_pkg->vor1_freq, 100.0, 6, "double",
         final_string);
string_gen(final_string, nav1_X1, nav1_Y1, size, &length,
           conv);
p = insert(p, length, conv);

/*   PUT UP VOR2   */

zero_pad(clock_pkg->vor2_freq, 100.0, 6, "double",
         final_string);
string_gen(final_string, nav2_X1, nav2_Y1, size, &length,
           conv);
p = insert(p, length, conv);

/* PUT UP TIMER */
```

```
        timer(page_number, clock_pkg, alarm_pkg, &length, conv);
        p = insert(p, length, conv);

        SCREEN[p++] = 0x8E00;
        SCREEN[p++] = 0xFFFF;

        times = 0;
        error = 1;
        while (error != 0)
            {
            times++;
            error = SEND_SCREEN( );
            }

/*      if(times != 1)
          printf("CK US times = %d\n", times);        */

}       /* end of dat_pg_dynamic */
```

```
/*************************************************************
 *
 *  SOURCE FILE:      nav_pg_s.c
 *
 *
 *  FUNCTION:         nav_pg_static()
 *
 *
 *  DESCRIPTION:      This routine generates the static data
 *                    for the NAV PAGE.  It is only called
 *                    once when the SYSTEM SWITCH is turned
 *                    on.
 *
 *
 *  DOCUMENTATION
 *  FILES:            None.
 *
 *
 *  ARGUMENTS:        None.
 *
 *
 *  RETURN:           None.
 *
 *
 *  FUNCTIONS
 *  CALLED:           UPDATE_SCREEN()
 *                    strcpy()
 *                    string_gen()
 *                    insert()
 *                    box()
 *                    line()
 *                    arc_circ()
 *
 *
 *  AUTHOR:           CHUCK ROBERTSON
 *
 *
 *  DATE CREATED:     15Jan87      Version 1.0
 *
 *
 *  REVISIONS:        None.
 *
 *
 *************************************************************/
#include<stdlib.h>
#include<string.h>
#include<stdio.h>
#include "navpg_xy.h"
#include "data_str.h"
```

159

```c
void nav_pg_static()
{

    unsigned short conv[300];           /* conversion array */
    unsigned short address_ptr;         /* address pointer */
    unsigned short line_char;           /* line characteristic */
    unsigned short internal_jmp;        /* HP internal jump */
    int XO, YO;                         /* plot coordinates */
    int p;                              /* conversion array
                                           index */
    int length;                         /* insertion length */
    int i;                              /* loop counter */
    int total_sent;                     /* total words sent to
                                           vector memory       */
    int error;                          /* masm return error */
    char final_string[30];              /* string arrays */
    char cat_string[10];
    double size;                        /* character size */

    int  insert();
    void string_gen();
    void box();
    void line();
    void arc_circ();
    void plane();

    address_ptr = 0xC300;       /* point at 0x0300 word in
                                   vector memory        */
    line_char = 0x7818;         /* set line characteristics */
    size = 1.5;                 /* set character size */

    p=0;
    SCREEN[p++] = address_ptr;
    SCREEN[p++] = line_char;

        /* PUT UP AIRSPEED */

        strcpy(final_string, "AIRSPD:");
        string_gen(final_string, airspeed_XO, airspeed_YO,
                   size, &length, conv);
        p = insert(p, length, conv);

        /* PUT UP GROUNDSPEED */

        strcpy(final_string, "GNDSPD:");
        string_gen(final_string, gndspeed_XO, gndspeed_YO,
                   size, &length, conv);
        p = insert(p, length, conv);

        /* PUT UP TRACK */
```

160

```c
strcpy(final_string, "TRK:");
string_gen(final_string, track_X0, track_Y0,
           size, &length, conv);
p = insert(p, length, conv);

/* PUT UP ALTITUDE */

strcpy(final_string, "ALT:");
string_gen(final_string, altitude_X0, altitude_Y0,
           size, &length, conv);
p = insert(p, length, conv);

/* PUT UP DME */

strcpy(final_string, "DME:   ");
string_gen(final_string, dme_X0, dme_Y0, size,
           &length, conv);
p = insert(p, length, conv);

box(dme_X0, dme_Y0, size, length-2, &length, conv);
p = insert(p, length, conv);

/* PUT UP PLANE */

plane(plane_X0, plane_Y0, &length, conv);
p = insert(p, length, conv);

/* PUT UP STATIC ARC OF COMPASS */

arc_circ(compass_arc_X0, compass_arc_Y0,
         compass_arc_init_angle,
         compass_arc_end_angle,
         compass_arc_radius, &length, conv);
p = insert(p, length, conv);

/* PUT UP HEADING BOX AND POINTER */

box(heading_box_X0, heading_box_Y0, size,
    heading_box_no_char, &length, conv);
p = insert(p, length, conv);

line(heading_ptr_line1_X0, heading_ptr_line1_Y0,
     heading_ptr_line1_X1, heading_ptr_line1_Y1,
     &length, conv);
p = insert(p, length, conv);

line(heading_ptr_line2_X0, heading_ptr_line2_Y0,
     heading_ptr_line2_X1, heading_ptr_line2_Y1,
     &length, conv);
p = insert(p, length, conv);
```

161

```
    SCREEN[p++] = 0x8900;
    SCREEN[p++] = 0xC900;
    SCREEN[p++] = 0x0000;
    SCREEN[p++] = 0x8FFF;
    SCREEN[p++] = 0xFFFF;

    error = 1;
    while (error != 0)
        error = SEND_SCREEN( );

}   /* end of nav_pg_static */
```

```
/****************************************************************
 *
 *   SOURCE FILE:      nav_pg_d.c
 *
 *
 *   FUNCTION:         nav_pg_dynamic()
 *
 *
 *   DESCRIPTION:      This function generates all the
 *                     dynamic data for the NAV PAGE. It
 *                     retrieves raw data from the data
 *                     package and makes necessary
 *                     conversions. Presently it is
 *                     displaying the following:
 *
 *                     1) airspeed
 *                     2) ground speed
 *                     3) ground track
 *                     4) dme
 *                     5) plane heading
 *                     6) compass
 *                     7) vortac's
 *                     8) ndb's
 *                     9) alarm conditions
 *                     10) altitude
 *
 *
 *   DOCUMENTATION
 *   FILES:            None.
 *
 *
 *   ARGUMENTS:        clock_pkg : pointer to clock package
 *                     alarm_pkg : pointer to alarm package
 *
 *
 *   RETURN:           None.
 *
 *
 *   FUNCTIONS
 *   CALLED:           GET_DATA_PACKAGE()
 *                     TOGGLE_ALARM_SWITCH()
 *                     UPDATE_SCREEN()
 *                     insert()
 *                     string_gen()
 *                     get_heading()
 *                     clim_arrow()
 *                     box()
 *                     zero_pad()
 *                     vortac()
 *                     ndb()
```

163

```
*                       line()
*                       compass()
*                       heading_and_bearing()
*                       airspeed()
*                       altitude()
*                       dme()
*                       get_ndb_plot_angle()
*                       timer()
*
*
*   AUTHOR:         CHUCK ROBERTSON
*
*
*   DATE CREATED:   09Feb87      Version 1.0
*
*
*   REVISIONS:      None.
*
*
***********************************************************/
#include "data_str.h"
#include "navpg_xy.h"
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<stdio.h>


void nav_pg_dynamic(clock_pkg,alarm_pkg)

CLOCK_PKG *clock_pkg;
ALARM_PKG *alarm_pkg;

{
    unsigned short conv[300];       /* conversion array */
    int p;                          /* conversion array
                                       index */
    int length;                     /* insertion length */
    int i;                          /* loop counter */
    int plane_heading_deg;          /* plane heading */
    int times;                      /* request counter */
    int radix;                      /* conversion radix */
    int int_dme;                    /* integer dme value */
    int X, Y;                       /* plot coordinates */
    int error;                      /* masm error return */
    int page_number = 2;            /* page being
                                       displayed */

    char final_string[20];          /* conversion string
                                        arrays */


                            164
```

```c
        char cat_string[20];

        int     insert();
        void    string_gen();
        void    box();
        void    get_heading();
        void    vortac();
        void    ndb();
        void    line();
        void    compass();
        void    heading_and_bearing();
        void    zero_pad();
        void    airspeed();
        void    altitude();
        void    dme();
        void    get_ndb_plot_angle();
        void    timer();

        double size;                    /* character size       */

        double real_delta_x;            /* converted delta x and */
        double real_delta_y;            /* delta y relative to
                                           VOR   */

        double plane_delta_x;           /* converted delta x and */
        double plane_delta_y;           /* delta y relative to
                                           plane */

        double theta_rad;               /* calculation angle
                                           degrees */
        double theta_deg;               /* calculation angle
                                           radians */

        double screen_scale;
        double PI;
        double DEGREE_TO_RAD;

        double vor_theta_rad;           /* vor plot angle in
                                           radians */
        double vor_theta_deg;           /* vor plot angle in
                                           degrees */
        double radius;                  /* plot radius          */
        double real_dme;                /* real dme value       */
        double vor_distance_scale;      /* scale for vor
                                           distance    */

        double ndb_theta_deg;           /* ndb plot angle in
                                           degrees */
        double ndb_theta_rad;           /* ndb plot angle in
                                           radians */
```

```
      times = O;
      error = 1;
      while (error != O)
          {
          times++;
          error = GET_DATA_PACKAGE( );
          }

/*    if (times != 1)
          printf("NV DP times = %d\n",times); */

      screen_scale = 9.5/12.5;
      PI = 3.14159265;
      DEGREE_TO_RAD = (2 * PI / 360.0);
      size = 1.5;

      p=O;
      SCREEN[p++] = OxC900;
      SCREEN[p++] = Ox7818;                      /* set line size */

      /* PUT UP AIRSPEED */

      airspeed(page_number, clock_pkg, alarm_pkg,
              airspeed_X1, airspeed_Y1,
              &length, conv);
      p = insert(p, length, conv);

      /* PUT UP GROUNDSPEED */

      /* yet to be implemented */

      /* PUT UP TRACK */

      /* yet to be implemented */

      /* PUT UP ALTITUDE */

      altitude(clock_pkg, alarm_pkg, altitude_X1, altitude_Y1,
              &length, conv);
      p = insert(p, length, conv);

      /* PUT UP DME */

      dme(clock_pkg, alarm_pkg, dme_X1, dme_Y1, &length, conv);
      p = insert(p, length, conv);

      /* PUT UP COMPASS AND HEADING */

      get_heading(&plane_heading_deg);
```

166

```
zero_pad((double)(plane_heading_deg),100.0,6,
        "int",final_string);
string_gen(final_string,heading_X0,heading_Y0,size,
        &length,conv);
p = insert(p,length,conv);

compass(compass_arc_X0,compass_arc_Y0,plane_heading_deg,
        compass_arc_radius,&length,conv);
p = insert(p,length,conv);

/* PUT UP VORTAC */

/* calculate real dme value */

real_dme = (double)((data_pkg.DME - 2.0) * 0.207);

/* plot vortac if plane is within 20 miles */

if(real_dme <= 30.0)
   {

      /* calculate delta x and y relative to vor */

      real_delta_x = (double)
                     ((data_pkg.DELTA_X - 126.0)
                        * 0.392);
      real_delta_y = (double)
                     ((data_pkg.DELTA_Y - 126.0)
                        * 0.415);

      /* calculate delta x and y relative to plane */

      plane_delta_x = -(real_delta_x);
      plane_delta_y = -(real_delta_y);

      /* CALCULATE POSITION OF VOR RELATIVE TO PLANE */

      if( (plane_delta_y != 0.0)
         && (plane_delta_x != 0.0) )
         {
            /* calculate vor angle relative to plane
               position                             */

            theta_rad = atan2(plane_delta_y,
                              plane_delta_x);

            /* add in the planes heading */

            vor_theta_rad = theta_rad +
                            (plane_heading_deg *
```

```
                        DEGREE_TO_RAD);

        vor_theta_deg = vor_theta_rad /
                                DEGREE_TO_RAD;

        if(vor_theta_deg >= 360.0)
            vor_theta_deg = vor_theta_deg - 360.0;

    }
else
    vor_theta_deg = 0.0;

    /* calculate new vor_theta_rad */

    vor_theta_rad = vor_theta_deg * DEGREE_TO_RAD;

    /* calculate plot radius */

    if(real_dme <= 0.0)
        radius = 0.0;
    else
        radius = 406.2 * log10(real_dme);

    /* calculate vortac plot coordinates */

    X = (int)(radius * cos(vor_theta_rad)
                * screen_scale);
    Y = (int)(radius * sin(vor_theta_rad));

    /* plot vortac */

    vortac(X + plane_XO, Y + plane_YO, &length, conv);
    p = insert(p, length, conv);

    /* put up vortac line */

    line(plane_XO, plane_YO, plane_XO + X,
        plane_YO + Y, &length, conv);
    p = insert(p, length, conv);

    /* put up bearing and heading if radius
        > 400 */

    if(radius >= 400.0)
    (

    /* put vortac heading and bearing */

    heading_and_bearing(plane_XO, plane_YO,
                        plane_heading_deg,
```

```
                                    vor_theta_deg,
                                    radius, &length, conv);
            p = insert(p, length, conv);

            }

    }

/* PUT UP NDB */

get_ndb_plot_angle(&ndb_theta_deg);

ndb_theta_rad = ndb_theta_deg * DEGREE_TO_RAD;

radius = 700.0;

/* calculate ndb plot coordinates */

X = (int)(radius * cos(ndb_theta_rad) * screen_scale);
Y = (int)(radius * sin(ndb_theta_rad));

/* plot ndb */

ndb(X + plane_XO, Y + plane_YO, &length, conv);
p = insert(p, length, conv);

/* put up ndb line */

line(plane_XO, plane_YO, plane_XO + X, plane_YO + Y,
     &length, conv);
p = insert(p, length, conv);

/* put up ndb heading and bearing */

heading_and_bearing(plane_XO, plane_YO, plane_heading_deg,
                    ndb_theta_deg, radius, &length, conv);
p = insert(p, length, conv);

/* CALL TIMER */

timer(page_number, clock_pkg, alarm_pkg, &length, conv);
p = insert(p, length, conv);

SCREEN[p++] = 0x8E00;
SCREEN[p++] = 0xFFFF;

times = 0;
error = 1;
while (error != 0)
     {
```

169

```
        times++;
        error = SEND_SCREEN( );
        }

/*    if (times != 1)
        printf("NV US times = %d\n",times); */

}   /* end of nav_pg_dynamic */
```

```
/****************************************************************
 *
 *   SOURCE FILE:      ils_pg_s.c
 *
 *
 *   FUNCTION:         ils_pg_static()
 *
 *
 *   DESCRIPTION:      This routine generates the static
 *                     data for the ILS PAGE.  It is only
 *                     called once when the SYSTEM SWITCH
 *                     is turned on.
 *
 *
 *   DOCUMENTATION
 *   FILES:            None.
 *
 *
 *   ARGUMENTS:        None.
 *
 *
 *   RETURN:           None.
 *
 *
 *   FUNCTIONS
 *   CALLED:           UPDATE_SCREEN()
 *                     string_gen()
 *                     insert()
 *                     arc_circ()
 *                     line()
 *
 *
 *   AUTHOR:           CHUCK ROBERTSON
 *
 *
 *   DATE CREATED:     16Feb87     Version 1.0
 *
 *
 *   REVISIONS:        None.
 *
 *
 ****************************************************************/
#include<stdlib.h>
#include<string.h>
#include<stdio.h>
#include "ilspg_xy.h"
#include "data_str.h"

void ils_pg_static()
{
```

171

```c
unsigned short conv[300];        /* conversion array */
unsigned short address_ptr;      /* address pointer */
unsigned short line_char;        /* line characteristic */
int p;                           /* conversion array
                                    index */
int length;                      /* insertion length */
int i;                           /* loop counter */
int total_sent;                  /* total words sent to
                                    vector memory      */
int error;                       /* masm error return */
char final_string[30];           /* string arrays */
char cat_string[10];
double size;                     /* character size */

void string_gen();
int  insert();
void box();
void arc_circ();
void line();

size = 1.5;                 /* set character size */
address_ptr = 0xC500;       /* point at 0x500 word in
                               vector memory       */
line_char = 0x7818;         /* set line characteristics */

p=0;
SCREEN[p++] = address_ptr;
SCREEN[p++] = line_char;

    /* PUT UP ALTITUDE */

    strcpy(final_string, "ALT:      ");
    string_gen(final_string, altitude_XO, altitude_YO,
               size, &length, conv);
    p = insert(p, length, conv);

    box(altitude_XO, altitude_YO, size, length-2,
        &length, conv);
    p = insert(p, length, conv);

    /* PUT UP DECISION HEIGHT */

    strcpy(final_string, "DH:");
    string_gen(final_string, mda_dh_XO, mda_dh_YO, size,
               &length, conv);
    p = insert(p, length, conv);

    /* PUT UP DME */

    strcpy(final_string, "DME:");
```

172

```
            string_gen(final_string, dme_XO, dme_YO, size,
                       &length, conv);
            p = insert(p, length, conv);

            /* PUT UP STATIC ARC OF COMPASS */

            arc_circ(compass_arc_XO, compass_arc_YO,
                     compass_arc_init_angle,
                     compass_arc_end_angle,
                     compass_arc_radius, &length, conv);
            p = insert(p, length, conv);

            /* PUT UP HEADING BOX AND POINTER */

            box(heading_box_XO, heading_box_YO, size,
                heading_box_no_char, &length, conv);
            p = insert(p, length, conv);

            line(heading_ptr_line1_XO, heading_ptr_line1_YO,
                 heading_ptr_line1_X1, heading_ptr_line1_Y1,
                 &length, conv);
            p = insert(p, length, conv);

            line(heading_ptr_line2_XO, heading_ptr_line2_YO,
                 heading_ptr_line2_X1, heading_ptr_line2_Y1,
                 &length, conv);
            p = insert(p, length, conv);

    SCREEN[p++] = 0x8C00;
    SCREEN[p++] = 0xCC00;
    SCREEN[p++] = 0x0000;
    SCREEN[p++] = 0x8FFF;
    SCREEN[p++] = 0xFFFF;

    error = 1;
    while (error != 0)
        error = SEND_SCREEN( );

}   /* end of ils_pg_static */
```

```
/************************************************************
 *
 *   SOURCE FILE:      ils_pg_d.c
 *
 *
 *   FUNCTION:         ils_pg_dynamic()
 *
 *
 *   DESCRIPTION:      This function generates all the
 *                     dynamic data for the ILS PAGE. It
 *                     retrieves raw data from the data
 *                     package and makes necessary
 *                     conversions. Presently it is
 *                     displaying the following:
 *
 *                       1) altitude
 *                       2) decision height
 *                       3) plane heading
 *                       4) dme
 *                       5) ils cross hairs
 *                       6) runway
 *                       7) alarm conditions
 *
 *
 *   DOCUMENTATION
 *   FILES:            None.
 *
 *
 *   ARGUMENTS:        clock_pkg : pointer to clock package
 *                     alarm_pkg : pointer to alarm package
 *
 *
 *   RETURN:           None.
 *
 *
 *   FUNCTIONS
 *   CALLED:           GET_DATA_PACKAGE()
 *                     TOGGLE_ALARM_SWITCH()
 *                     UPDATE_SCREEN()
 *                     insert()
 *                     string_gen()
 *                     get_heading()
 *                     box()
 *                     zero_pad()
 *                     line()
 *                     runway()
 *                     altitude()
 *                     dme()
 *                     zero_pad()
 *                     ils_compass()
```

174

```
   *                      airspeed()
   *                      timer()
   *
   *
   *   AUTHOR:            CHUCK ROBERTSON
   *
   *
   *   DATE CREATED:      16Feb87      Version 1.0
   *
   *
   *   REVISIONS:         None.
   *
   *
   ************************************************************/
#include "data_str.h"
#include "ilspg_xy.h"
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<stdio.h>

void ils_pg_dynamic(clock_pkg, alarm_pkg)

CLOCK_PKG *clock_pkg;
ALARM_PKG *alarm_pkg;

{
    unsigned short conv[300];      /* conversion array */
    int     times;                 /* request times */
    int     p;                     /* array index */
    int     length;                /* insertion length */
    int     i;                     /* counter */
    int     radix;                 /* conversion */
    int     error;                 /* masm error return */
    int     page_number = 3;       /* page being displayed */

    int     plane_heading_deg;     /* plane heading */
    int     glideslope;            /* glideslope plot
                                      coord */
    int     course_deviation;      /* course_deviation
                                      coord */
    int     runway_hdg_deg;        /* runway heading */

    int     insert();
    void    string_gen();
    void    runway();
    void    box();
    void    line();
    void    get_heading();
    void    altitude();


                           175
```

```c
      void   dme();
      void   zero_pad();
      void   ils_compass();
      void   airspeed();
      void   timer();

      int    lower_X_bound;              /* plot boundries */
      int    upper_X_bound;
      int    lower_Y_bound;
      int    upper_Y_bound;
      int    middle_X;                   /* cross hair
                                            coordinates */
      int    middle_Y;
      int    lower_pegout_bound;         /* pegout bounds */
      int    upper_pegout_bound;
      int    left_pegout_bound;
      int    right_pegout_bound;

      unsigned char   peg_left;          /* sensitivity bounds */
      unsigned char   peg_up;
      unsigned char   peg_right;
      unsigned char   peg_down;

      double delta_x;                    /* plot divisions */
      double delta_y;

      double real_dme;                   /* real dme value */
      double MIN_runway_scale;           /* minimum runway scale */
      double MAX_runway_scale;           /* maximum runway scale */
      double MAX_plot_dme;               /* maximum plot dme */
      double delta_runway_scale;         /* scale multiplier */
      double runway_scale;               /* runway scale */
      double crab_angle_deg;             /* crab angle of runway */

      char   final_string[20];           /* conversion string
                                            arrays */
      char   cat_string[20];
      double size;                       /* character size */

      times = 0;
      error = 1;
      while (error != 0)
          {
          times++;
          error = GET_DATA_PACKAGE( );
          }

/*    if (times != 1)
          printf("ILS DP %d\n",times); */
```

176

```c
    size = 1.5;
    radix = 10;
    runway_hdg_deg = 280;
    MIN_runway_scale = 1.0;
    MAX_runway_scale = 5.0;
    MAX_plot_dme = 15.0;
    delta_runway_scale = (MAX_runway_scale -
                        MIN_runway_scale) /
                        MAX_plot_dme;

    p=0;
    SCREEN[p++] = 0xCC00;
    SCREEN[p++] = 0x7818;                    /* set line size */

    /* CHECK AIRSPEED */

    airspeed(page_number,clock_pkg, alarm_pkg,
            airspeed_X1, airspeed_Y1,
            &length, conv);
    p = insert(p, length, conv);

    /* PUT UP ALTITUDE */

    altitude(clock_pkg, alarm_pkg, altitude_X1, altitude_Y1,
            &length, conv);
    p = insert(p, length, conv);

    /* PUT UP DECISION HEIGHT */

    zero_pad((double)(clock_pkg->mda_dh), 1000.0, 6, "int",
            final_string);
    string_gen(final_string, mda_dh_X1, mda_dh_Y1, size,
            &length, conv);
    p = insert(p, length, conv);

    /* PUT UP COMPASS AND HEADING */

    get_heading(&plane_heading_deg);

    zero_pad((double)(plane_heading_deg), 100.0, 6,
            "int", final_string);
    string_gen(final_string, heading_X0, heading_Y0, size,
            &length, conv);
    p = insert(p, length, conv);

    ils_compass(compass_arc_X0, compass_arc_Y0,
                plane_heading_deg, compass_arc_radius,
                &length, conv);
    p = insert(p, length, conv);
```

177

```
/* PUT UP DME */

dme(clock_pkg, alarm_pkg, dme_X1, dme_Y1, &length, conv);
p = insert(p, length, conv);

real_dme = (double)((data_pkg.DME - 2.0) * 0.207);

if(real_dme < 0.0)
   real_dme = - real_dme;

/* PUT UP CDI */

/*    strcpy(final_string, "CDI:");
   itoa(data_pkg.COURSE_DEVIATION, cat_string, radix);
   strcat(final_string, cat_string);
   string_gen(final_string, 1500, 1850, size, &length, conv);
   p = insert(p, length, conv); */

/* PUT UP GLDSLP */

/*    strcpy(final_string, "GLDSLP:");
   itoa(data_pkg.GLIDESLOPE, cat_string, radix);
   strcat(final_string, cat_string);
   string_gen(final_string, 1500, 1750, size, &length, conv);
   p = insert(p, length, conv); */

/* IF PLANE IS CLOSE ENOUGH PLOT CROSSHAIRS */

if(real_dme <= 20.0)
   {

      /* CALCULATE RUNWAY SCALE */

      runway_scale = MAX_runway_scale -
                     real_dme * delta_runway_scale;

      /* CALCULATE BOUNDRIES FOR RUNWAY PLOT */

      lower_X_bound = (int)(75 * runway_scale);
      upper_X_bound = (int)(2048 - (75 *
                              runway_scale));
      middle_X = (int)(lower_X_bound +
                       (upper_X_bound -
                        lower_X_bound) / 2.0);

      lower_Y_bound = (int)(50 * runway_scale);
      upper_Y_bound = (int)(1850 -
                              (150 * runway_scale));
      middle_Y = (int)(lower_Y_bound +
                       (upper_Y_bound -
```

```
                        lower_Y_bound) / 2.0);

/* PUT UP INITIAL INDICATOR CROSSHAIRS */

if(real_dme > MAX_plot_dme)
    {

        line(middle_X, lower_Y_bound, middle_X,
            upper_Y_bound, &length, conv);
        p = insert(p, length, conv);

        line(lower_X_bound, middle_Y, upper_X_bound,
            middle_Y, &length, conv);
        p = insert(p, length, conv);

    }

/* IF PLANE IS CLOSE ENOUGH PLOT RUNWAY */

if(real_dme <= MAX_plot_dme)
    {

        /* ADD IN SENSITIVITY */

        if(real_dme >= 8.0)
            {
                peg_left = peg_up = 50;
                peg_right = peg_down = 205;
            }
        else if(real_dme >= 7.0)
            {
                peg_left = peg_up = 52;
                peg_right = peg_down = 202;
            }
        else if(real_dme >= 6.0)
            {
                peg_left = peg_up = 62;
                peg_right = peg_down = 191;
            }
        else if(real_dme >= 5.0)
            {
                peg_left = peg_up = 75;
                peg_right = peg_down = 181;
            }
        else if(real_dme >= 4.0)
            {
                peg_left = peg_up = 81;
                peg_right = peg_down = 174;
            }
        else if(real_dme >= 3.0)
```

179

```
            {
                peg_left = peg_up = 87;
                peg_right = peg_down = 165;
            }
        else if(real_dme >= 2.0)
            {
                peg_left = peg_up = 92;
                peg_right = peg_down = 158;
            }
        else if(real_dme >= 1.0)
            {
                peg_left = peg_up = 106;
                peg_right = peg_down = 147;
            }
        else if(real_dme >= 0.0)
            {
                peg_left = peg_up = 117;
                peg_right = peg_down = 135;
            }

        if(data_pkg.COURSE_DEVIATION < peg_left)
            data_pkg.COURSE_DEVIATION = peg_left;

        if(data_pkg.COURSE_DEVIATION > peg_right)
            data_pkg.COURSE_DEVIATION = peg_right;

        if(data_pkg.GLIDESLOPE < peg_up)
            data_pkg.GLIDESLOPE = peg_up;

        if(data_pkg.GLIDESLOPE > peg_down)
            data_pkg.GLIDESLOPE = peg_down;


        /* CALCULATE COORDINATES FOR RUNWAY */

        delta_x = (double)(
                    (upper_X_bound -
                     lower_X_bound) /
                    (peg_right - peg_left));

        delta_y = (double)(
                    (upper_Y_bound -
                     lower_Y_bound) /
                    (peg_down - peg_up));

        course_deviation

                = lower_X_bound +
                  (int)
                  ((data_pkg.COURSE_DEVIATION -
```

```
                        peg_left) * delta_x);

        glideslope

                = upper_Y_bound -
                  (int)
                  ((data_pkg.GLIDESLOPE - peg_up)
                   * delta_y);

        /* PUT UP PEG-OUT BOUNDARIES */

        lower_pegout_bound = (int)(lower_Y_bound +
                                   2 * delta_y);

        upper_pegout_bound = (int)(upper_Y_bound -
                                   2 * delta_y);

        left_pegout_bound = (int)(lower_X_bound +
                                  2 * delta_x);

        right_pegout_bound = (int)(upper_X_bound -
                                   2 * delta_x);

        line(left_pegout_bound, lower_pegout_bound,
             left_pegout_bound, upper_pegout_bound,
             &length, conv);
        p = insert(p, length, conv);

        line(left_pegout_bound, upper_pegout_bound,
             right_pegout_bound, upper_pegout_bound,
             &length, conv);
        p = insert(p, length, conv);

        line(right_pegout_bound, upper_pegout_bound,
             right_pegout_bound, lower_pegout_bound,
             &length, conv);
        p = insert(p, length, conv);

        line(right_pegout_bound, lower_pegout_bound,
             left_pegout_bound, lower_pegout_bound,
             &length, conv);
        p = insert(p, length, conv);

        /* PUT UP APPROACH CROSSHAIRS */

        line(middle_X, lower_pegout_bound,
             middle_X, upper_pegout_bound,
             &length, conv);
        p = insert(p, length, conv);
```

```
                line(left_pegout_bound, middle_Y,
                     right_pegout_bound, middle_Y,
                     &length, conv);
                p = insert(p, length, conv);

                /* CALCULATE CRAB ANGLE */

                crab_angle_deg = -(double)(
                                     (plane_heading_deg + 360)
                                     - (runway_hdg_deg
                                     % 360));

                if(crab_angle_deg <= 360.0)
                   crab_angle_deg = crab_angle_deg + 360.0;

                if(crab_angle_deg > 20.0)
                   crab_angle_deg = 20.0;

                if(crab_angle_deg < -20.0)
                   crab_angle_deg = -20.0;

                /* PLOT RUNWAY */

                runway(course_deviation, glideslope,
                       crab_angle_deg, runway_scale, &length,
                       conv);
                p = insert(p, length, conv);

              }

        }

     /* CHECK TIMER */

     timer(page_number, clock_pkg, alarm_pkg, &length, conv);
     p = insert(p, length, conv);

     SCREEN[p++] = 0x8E00;
     SCREEN[p++] = 0xFFFF;

     times = 0;
     error = 1;
     while (error != 0)
          {
          times++;
          error = SEND_SCREEN( );
          }

/*   if (times != 1)
         printf("ILS US times = %d\n", times); */

                          182
```

```
}    /* end of ils_pg_dynamic */
```

APPENDIX D (CONT.)

GENERAL-PURPOSE TOOLS

```
/******************************************************************
 *
 *   SOURCE FILE:      insert.c
 *
 *
 *   FUNCTION:         insert()
 *
 *
 *   DESCRIPTION:      This function inserts the conversion
 *                     array into the screen array. The
 *                     length of the conversion array is
 *                     passed to the routine.
 *
 *
 *   DOCUMENTATION
 *   FILES:            None.
 *
 *
 *   ARGUMENTS:        p      : index for the SCREEN array
 *                     length : length of the conversion
 *                              array
 *                     conv   : pointer to the conversion
 *                              array
 *
 *   RETURN:           p      : index for the SCREEN array
 *
 *
 *   FUNCTIONS
 *   CALLED:           None.
 *
 *
 *   AUTHOR:           CHUCK ROBERTSON
 *
 *
 *   DATE CREATED:     22Jan87      Version 1.0
 *
 *
 *   REVISIONS:        None.
 *
 *
 ******************************************************************/

insert(p, length, conv)

unsigned short p;
unsigned short length;
unsigned short conv[];

{
    int i;                           /* loop counter */
```

```
extern unsigned short SCREEN[];    /* screen data array */

for(i=0;i < length ;i++)
    SCREEN[p++] = conv[i];

return (p);

}   /* end of insert */
```

```
/****************************************************************
 *
 *   SOURCE FILE:      line.c
 *
 *
 *   FUNCTION:         line()
 *
 *
 *   DESCRIPTION:      This program generates the HP code to
 *                     put up a line between two points on
 *                     vector graphics display.
 *
 *
 *   DOCUMENTATION
 *   FILES:            None.
 *
 *
 *   ARGUMENTS:        XO,YO          : beginning coordinates
 *                                      for the line
 *                     X1,Y1          : ending coordinates
 *                                      for the line
 *                     length_ptr     : pointer to the length of
 *                                      conv[] array
 *                     conv           : pointer to converion
 *                                      array
 *
 *   RETURN:           None.
 *
 *
 *   FUNCTIONS
 *   CALLED:           None.
 *
 *
 *   AUTHOR:           CHUCK ROBERTSON
 *
 *
 *   DATE CREATED:     22Jan87      Version 1.0
 *
 *
 *   REVISIONS:        None.
 *
 *
 ****************************************************************/

void line(XO,YO,X1,Y1,length_ptr,conv)

unsigned short XO,YO;
unsigned short X1,Y1;
unsigned short *length_ptr;
unsigned short conv[];
```

```
{
    conv[0] = 0x0000 | X0;      /* plot x off */
    conv[1] = 0x1000 | Y0;      /* plot y off */
    conv[2] = 0x0800 | X1;      /* plot x on */
    conv[3] = 0x1800 | Y1;      /* plot y on */

    *length_ptr = 4;

}   /* end of line */
```

```
/***************************************************************
 *
 *   SOURCE FILE:        arc_circ.c
 *
 *
 *   FUNCTION:           arcd_circ()
 *
 *
 *   DESCRIPTION:        This routine generates the HP code to
 *                       put up an arc from 0 to 360 degrees
 *                       with plot origin at center of arc.
 *
 *
 *   DOCUMENTATION
 *   FILES:              None.
 *
 *
 *   ARGUMENTS:          x_center          : x plot coordinate
 *                                           for arc center
 *                       y_center          : y plot coordinate
 *                                           for arc center
 *                       start_angle_deg : initial angle
 *                       end_angle_deg   : final angle
 *                       radius          : radius of arc
 *                       length_ptr      : pointer to length of
 *                                           conv[] array
 *                       conv              : pointer to
 *                                           conversion array
 *
 *   RETURN:             None.
 *
 *
 *   FUNCTIONS
 *   CALLED:             None.
 *
 *
 *   AUTHOR:             CHUCK ROBERTSON
 *
 *
 *   DATE CREATED:       25Jan87       Version 1.0
 *
 *
 *   REVISIONS:          None.
 *
 ***************************************************************/
#include<math.h>
#include<stdio.h>

void arc_circ(x_center,y_center,start_angle_deg,
```

```c
                  end_angle_deg, radius, length_ptr, conv)

unsigned short conv[];
unsigned short *length_ptr;
int x_center, y_center;
int radius;
double start_angle_deg;
double end_angle_deg;

{
    int     X, Y;                   /* plot coordinate
                                       variables      */
    int     final_delta_cnt;        /* iteration number */
    int     delta_count;            /* loop counter */
    int     i;                      /* loop counter */
    int     index;                  /* conversion index */

    double PI;
    double DEGREE_TO_RAD;
    double start_angle_rad;         /* beginning angle in
                                       radians         */
    double end_angle_rad;           /* ending angle in radians */
    double delta_theta_deg;         /* subinterval of total angle
                                       in degrees */
    double delta_theta_rad;         /* subinterval of total angle
                                       in radians */
    double theta;                   /* calculation angle */
    double total_angle;             /* total arc angle */
    double screen_scale;            /* screen scale for X
                                       direction */

    PI = 3.14159265;
    screen_scale = 9.5/12.5;
    DEGREE_TO_RAD = (2 * PI) / 360.0;
    index = 0;

    /* convert beginning and ending angles to radians */

    start_angle_rad = start_angle_deg * DEGREE_TO_RAD;
    end_angle_rad = end_angle_deg * DEGREE_TO_RAD;

    /* determine angle increment based on radius */

    if(radius <= 100)
      delta_theta_deg = 12.0;
    else if((radius > 100) && (radius <= 200))
      delta_theta_deg = 11.25;
    else if((radius > 200) && (radius <= 300))
      delta_theta_deg = 10.0;
    else if((radius > 300) && (radius <= 400))
```

190

```c
    delta_theta_deg = 9.0;
 else if((radius > 400) && (radius <= 500))
    delta_theta_deg = 8.0;
 else if((radius > 500) && (radius <= 600))
    delta_theta_deg = 7.2;
 else if((radius > 600) && (radius <= 700))
    delta_theta_deg = 6.0;
 else if((radius > 800) && (radius <= 900))
    delta_theta_deg = 5.0;
 else if((radius > 900) && (radius <= 1000))
    delta_theta_deg = 4.0;
 else if((radius > 1000) && (radius <= 1200))
    delta_theta_deg = 3.0;
 else if((radius > 1200) && (radius <= 1400))
    delta_theta_deg = 2.0;
 else if((radius > 1400) && (radius <= 2000))
    delta_theta_deg = 1.0;
 else printf("INVALID RADIUS INPUTED\n");

/* convert to radians */

delta_theta_rad = delta_theta_deg * DEGREE_TO_RAD;

/* initialize calculation angle */

theta = start_angle_rad;

/* calculate total arc angle */

total_angle = end_angle_deg - start_angle_deg;

/* calculate iteration # */

final_delta_cnt = (int)(total_angle/delta_theta_deg);

/* calculate initial coordinates */

X = (int)(radius * cos(theta) * screen_scale);
Y = (int)(radius * sin(theta));

conv[index++] = 0x0000 | (X + x_center); /* plot x
                                            off     */
conv[index++] = 0x1000 | (Y + y_center); /* plot y
                                            off     */

/* calculate rest of coordinates */

for(delta_count = 1; delta_count <= final_delta_cnt
    ; delta_count ++)
```

191

```c
    {
        theta = theta + delta_theta_rad;

        X = (int)(radius * cos(theta) * screen_scale);
        Y = (int)(radius * sin(theta));

        conv[index++] = 0x0000 | (X + x_center);
        conv[index++] = 0x1800 | (Y + y_center);

    }

*length_ptr = index;

} /* end arc_circ */
```

```c
/*****************************************************************
 *
 *   SOURCE FILE:        str_gen.c
 *
 *
 *   FUNCTION:           string_gen()
 *
 *
 *   DESCRIPTION:        This routine generates the HP code to
 *                       put a string at a certain location on
 *                       the vector graphics display.
 *
 *
 *   DOCUMENTATION
 *   FILES:              None.
 *
 *
 *   ARGUMENTS:          XO, YO      : coordinates for the
 *                                     string
 *                       length_ptr  : pointer to the length of
 *                                     conv[] array
 *                       size        : size of the character to
 *                                     be generated
 *                       conv        : pointer to converion
 *                                     array
 *                       string      : pointer to string array
 *   RETURN:             None.
 *
 *
 *   FUNCTIONS
 *   CALLED:             None.
 *
 *
 *   AUTHOR:             CHUCK ROBERTSON
 *
 *
 *   DATE CREATED:       22Jan87     Version 1.0
 *
 *
 *   REVISIONS:          None.
 *
 *
 *****************************************************************/
#include<string.h>
#include<stdio.h>

void string_gen(string, XO, YO, size, length_ptr, conv)
```

193

```c
unsigned short XO, YO;
unsigned short *length_ptr;
unsigned short conv[];
double size;
char string[];

{
    int i;                  /* loop counter */

    /* convert coordinates to hp code */

    conv[0] = 0x0000 | XO;  /* plot x off */
    conv[1] = 0x1000 | YO;  /* plot y off */

    /* convert size to hp code */

    if(size == 1.0)
        conv[2] = 0x4100;
    else if(size == 1.5)
        conv[2] = 0x4900;
    else if(size == 2.0)
        conv[2] = 0x5100;
    else if(size == 2.5)
        conv[2] = 0x5900;
    else printf("INVALID SIZE INPUTED\n");

    /* find string length */

    *length_ptr = strlen(string);

    /* add 2 to length pointer for XO and YO code */

    *length_ptr = *length_ptr + 2;

    /* convert string to hp code */

    conv[2] = conv[2] | string[0];

    for(i=3 ; i < *length_ptr ; i++)
        conv[i] = string[i-2] | 0x4000;

}   /* end of string_gen */
```

194

```
/ ****************************************************************
 *
 *   SOURCE FILE:      box.c
 *
 *
 *   FUNCTION:         box()
 *
 *
 *   DESCRIPTION:      This function generates the code to
 *                     put up a box on the vector graphics
 *                     display around a given word at a
 *                     given location.
 *
 *
 *   DOCUMENTATION
 *   FILES:            None.
 *
 *
 *   ARGUMENTS:        XO, YO       : plot coordinates for
 *                                    plane
 *                     size         : size of the characters
 *                     no_char      : number of characters to
 *                                  :  be boxed
 *                     length_ptr   : pointer to length of
 *                                    conv[] array
 *                     conv         : pointer to conversion
 *                                    array
 *
 *
 *   RETURN:           None.
 *
 *
 *   FUNCTIONS
 *   CALLED:           None.
 *
 *
 *   AUTHOR:           CHUCK ROBERTSON
 *
 *
 *   DATE CREATED:     25Jan87      Version 1.0
 *
 *
 *   REVISIONS:        None.
 *
 *
 ****************************************************************/
#include<stdio.h>

void box(XO, YO, size, no_char, length_ptr, conv)
```

```c
unsigned short conv[];
unsigned short *length_ptr;
unsigned short XO,YO;
unsigned short no_char;
double size;

{   int X,Y;                /* coordinate variables */
    int width;              /* character width */
    int height;             /* character height */

    if(size == 1.0)
       { width = 36;
         height = 36; }
    else if(size == 1.5)
       { width = 54;
         height = 54; }
    else if(size == 2.0)
       { width = 72;
         height = 72; }
    else if(size == 2.5)
       { width = 90;
         height = 90; }
    else printf("invalid size inputed\n");

    /* move to initial coordinates and generate box
       coordinates                                   */

    X = XO - 10;
    Y = YO - 10;

    conv[0] = 0x0000 | X;   /* plot x off */
    conv[1] = 0x1000 | Y;   /* plot y off */

    X = X + 20 + no_char * width;
    conv[2] = 0x0000 | X;
    conv[3] = 0x1800 | Y;

    Y = Y + 20 + height;
    conv[4] = 0x0000 | X;
    conv[5] = 0x1800 | Y;

    X = X - 20 - no_char * width;
    conv[6] = 0x0000 | X;
    conv[7] = 0x1800 | Y;

    Y = Y - 20 - height;
    conv[8] = 0x0000 | X;
    conv[9] = 0x1800 | Y;

    *length_ptr = 10;
```

```
}   /* end of box */
```

```
/***************************************************************
 *
 *  SOURCE FILE:     zero_pad.c
 *
 *
 *  FUNCTION:        zero_pad()
 *
 *
 *  DESCRIPTION:     This function zero pads a given
 *                   integer or double given the number to
 *                   be padded, the pad limit, the
 *                   precision, and type of number to be
 *                   padded.
 *
 *
 *  DOCUMENTATION
 *  FILES:           None.
 *
 *
 *  ARGUMENTS:       pad_value   : value to be padded
 *                   pad_limit   : zero padding limit
 *                   precision   : precision desired
 *                   type_string : type of number to be
 *                                 padded
 *                   pad_string  : pointer to conversion
 *                                 array
 *
 *
 *  RETURN:          None.
 *
 *
 *  FUNCTIONS
 *  CALLED:          strcpy()
 *                   strcmp()
 *                   strcat()
 *                   itoa()
 *                   gcvt()
 *                   printf()
 *
 *
 *  AUTHOR:          Chuck Robertson
 *
 *
 *  DATE CREATED:    25Feb87     Version 1.0
 *
 *
 *  REVISIONS:       None.
 *
 *
 ***************************************************************/
```

```c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

void zero_pad(pad_value, pad_limit, precision, type_string,
                pad_string)

char pad_string[20];
double pad_value;
double pad_limit;
int precision;
char type_string[];

{
    char   cat_string[19];
    int    radix = 10;

    /* start pad_string with EOL for strcat() to detect */

    pad_string[0] = '\0';

    /* take care of negative case */

    if(pad_value < 0.0)
        {
            strcpy(pad_string, "-");
            pad_value = pad_value * -1.0;
        }

    /* determine if padding is necessary */

    if( ((pad_limit / 10.0) <= pad_value) &&
        (pad_value < pad_limit) )
        strcpy(pad_string, "0");
    else if( ((pad_limit / 100.0) <= pad_value ) &&
            (pad_value < (pad_limit / 10.0)) )
        strcpy(pad_string, "00");
    else if( ((pad_limit / 1000.0) <= pad_value ) &&
            (pad_value < (pad_limit / 100.0)) )
        strcpy(pad_string, "000");

    if( (pad_value == 0.0) && (pad_limit >= 1000.0) )
        strcpy(pad_string, "000");
    else if( (pad_value == 0.0) && (pad_limit >= 100.0) )
        strcpy(pad_string, "00");
    else if( (pad_value == 0.0) && (pad_limit >= 10.0) )
        strcpy(pad_string, "0");

    /* convert pad_value to a character string */
```

199

```c
    if( strcmp(type_string, "int") == 0)
        itoa((int)(pad_value), cat_string, radix);
    else if( strcmp(type_string, "double") == 0)
        gcvt(pad_value, precision, cat_string);
    else
        printf("ILLEGAL TYPE INPUTED IN ZERO PAD\n");

    /* append pad_value to pad_string */

    strcat(pad_string, cat_string);

} /* end of zero_pad */
```

```
/****************************************************************
 *
 *  SOURCE FILE:      altitude.c
 *
 *
 *  FUNCTION:         altitude()
 *
 *
 *  DESCRIPTION:      This routine retrieves the altitude
 *                    value from the data package and
 *                    converts it into feet.  It then
 *                    converts this value into a zero-
 *                    padded plot string.
 *
 *
 *  DOCUMENTATION
 *  FILES:            None.
 *
 *
 *  ARGUMENTS:        clock_pkg      : pointer to clock
 *                                     package
 *                    alarm_pkg      : pointer to alarm
 *                                     package
 *                    XO,YO          : plot coordinates
 *                    length_ptr     : pointer to length of
 *                                     conversion array
 *                    altitude_conv  : pointer to conversion
 *                                     array
 *
 *
 *  RETURN:           None.
 *
 *
 *  FUNCTIONS
 *  CALLED:           zero_pad()
 *                    string_gen()
 *
 *
 *  AUTHOR:           CHUCK ROBERTSON
 *
 *
 *  DATE CREATED:     23Mar87        Version 1.0
 *
 *
 *  REVISIONS:        04Apr87        Version 1.1
 *                                   Added alarm capability.
 *
 ****************************************************************/
#include <stdlib.h>
```

```c
#include <string.h>
#include <stdio.h>
#include "data_str.h"      /* data package structure
                              declaration. */


void altitude(clock_pkg,alarm_pkg,X0,Y0,length_ptr,
              altitude_conv)

CLOCK_PKG *clock_pkg;
ALARM_PKG *alarm_pkg;
int X0,Y0;
int *length_ptr;
unsigned short altitude_conv[];

{
    unsigned short string_conv[30];
    int altitude;
    int string_length;
    int index = 0;
    int i;
    char final_string[20];
    double size1_5 = 1.5;

    void string_gen();
    void zero_pad();

    altitude = (int)((data_pkg.ALTITUDE - 15) * 29.7);

    if(altitude < 0)
       altitude = 0;

    zero_pad((double)(altitude),1000.0,6,"int",
             final_string);

    string_gen(final_string,X0,Y0,size1_5,&string_length,
               string_conv);

    for(i=0;i<string_length;i++)
        altitude_conv[index + i] = string_conv[i];

    index = index + string_length;

    /* CHECK MDA_DH ALARM CONDITION */

    /* IF ALTITUDE IS APPROACHING MDA_DH SOUND ALARM */

    if((alarm_pkg->mda_dh_alarm_flag == ALARM_OFF)
      && (altitude <= (clock_pkg->mda_dh + 50))
         && (alarm_pkg->mda_dh_enable_flag == ENABLED))
```

202

```
        (
        alarm_pkg->mda_dh_alarm_flag = ALARM_ON;

        if(alarm_pkg->alarm_status_flag == ALARM_OFF)
            (
            TOGGLE_ALARM_SWITCH();
            alarm_pkg->alarm_status_flag = ALARM_ON;
            )
        )

   if((alarm_pkg->mda_dh_alarm_flag == ALARM_ON)
     && (altitude > (clock_pkg->mda_dh + 50))
     && (alarm_pkg->mda_dh_enable_flag == ENABLED))
      (
      alarm_pkg->mda_dh_alarm_flag = ALARM_OFF;

      if((alarm_pkg->alarm_status_flag == ALARM_ON) &&
        (alarm_pkg->assigned_altitude_alarm_flag ==
         ALARM_OFF)
        && (alarm_pkg->airspeed_alarm_flag == ALARM_OFF)
        && (alarm_pkg->time_out_alarm_flag == ALARM_OFF))
         (
         TOGGLE_ALARM_SWITCH();
         alarm_pkg->alarm_status_flag = ALARM_OFF;
         )
      )

  /* IF ALARM IS ON DISPLAY MDA/DH APPROACHING MESSAGE */

  if((alarm_pkg->mda_dh_alarm_flag == ALARM_ON)
    && (alarm_pkg->mda_dh_enable_flag == ENABLED))
     (
        strcpy(final_string, ">>>MDA/DH<<<");
        string_gen(final_string, 680, 80,
                    size1_5, &string_length, string_conv);

        for(i=0;i<string_length;i++)
            altitude_conv[index + i] = string_conv[i];

        index = index + string_length;
     )

  /* CHECK ASSIGNED ALTITUDE ALARM CONDITION */

  /* IF ALTITUDE IS EXCEEDING ASSIGNED ALTITUDE LIMIT
     SOUND ALARM                                      */

  if( (alarm_pkg->assigned_altitude_alarm_flag ==
      ALARM_OFF)
    && !( (altitude <= (clock_pkg->assigned_altitude +
```

```
        100))
    && (altitude >= (clock_pkg->assigned_altitude -
       100)) )
    && (alarm_pkg->assigned_altitude_enable_flag ==
      ENABLED) )
     (
     alarm_pkg->assigned_altitude_alarm_flag = ALARM_ON;

     if(alarm_pkg->alarm_status_flag == ALARM_OFF)
        (
        TOGGLE_ALARM_SWITCH();
        alarm_pkg->alarm_status_flag = ALARM_ON;
        )
     )

  if( (alarm_pkg->assigned_altitude_alarm_flag ==
      ALARM_ON)
    && ( (altitude <= (clock_pkg->assigned_altitude +
       100))
    && (altitude >= (clock_pkg->assigned_altitude -
       100)) )
    && (alarm_pkg->assigned_altitude_enable_flag ==
      ENABLED) )
     (
     alarm_pkg->assigned_altitude_alarm_flag = ALARM_OFF;

     if((alarm_pkg->alarm_status_flag == ALARM_ON) &&
       (alarm_pkg->mda_dh_alarm_flag == ALARM_OFF)
       && (alarm_pkg->airspeed_alarm_flag == ALARM_OFF)
       && (alarm_pkg->time_out_alarm_flag == ALARM_OFF))
        (
        TOGGLE_ALARM_SWITCH();
        alarm_pkg->alarm_status_flag = ALARM_OFF;
        )
     )

  /* IF ALARM IS ON DISPLAY ASSIGNED ALTITUDE MESSAGE */

  if((alarm_pkg->assigned_altitude_alarm_flag == ALARM_ON)
    && (alarm_pkg->assigned_altitude_enable_flag ==
      ENABLED))
     (
        strcpy(final_string,">>ASSIGNED<<");
        string_gen(final_string,680,10,
                   size1_5,&string_length,string_conv);

        for(i=0;i<string_length;i++)
            altitude_conv[index + i] = string_conv[i];

        index = index + string_length;
```

204

```
        }

    *length_ptr = index;
} /* end altitude */
```

```
/****************************************************************
 *
 *   SOURCE FILE:      dme.c
 *
 *
 *   FUNCTION:         dme()
 *
 *
 *   DESCRIPTION:      This routine retrieves the dme value
 *                     from the data package and converts it
 *                     into nautical miles.  It then converts
 *                     this value into a zero-padded plot
 *                     string.
 *
 *
 *   DOCUMENTATION
 *   FILES:            None.
 *
 *
 *   ARGUMENTS:        clock_pkg       : pointer to clock
 *                                       package
 *                     alarm_pkg       : pointer to alarm
 *                                       package
 *                     XO, YO          : plot coordinates
 *                     length_ptr      : pointer to length of
 *                                       conversion array
 *                     dme_conv        : pointer to conversion
 *                                       array
 *
 *
 *   RETURN:           None.
 *
 *
 *   FUNCTIONS
 *   CALLED:           zero_pad()
 *                     string_gen()
 *
 *
 *   AUTHOR:           CHUCK ROBERTSON
 *
 *
 *   DATE CREATED:     23Mar87    Version 1.0
 *
 *
 *   REVISIONS:        None.
 *
 *
 ****************************************************************/
#define  ALARM_OFF   0
#define  ALARM_ON    1
```

```c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "data_str.h"    /* data package structure
                            declaration.          */

extern int TOGGLE_ALARM_SWITCH( );

void dme(clock_pkg, alarm_pkg, XO, YO, length_ptr, dme_conv)

CLOCK_PKG *clock_pkg;
ALARM_PKG *alarm_pkg;
int XO, YO;
int *length_ptr;
unsigned short dme_conv[];

{
    int dme;
    int length;
    char final_string[20];
    double size = 1.5;

    void string_gen();
    void zero_pad();

    dme = (int)((data_pkg.DME - 2.0) * 0.207);

    zero_pad((double)(dme), 10.0, 6, "int", final_string);
    string_gen(final_string, XO, YO, size, &length, dme_conv);

    *length_ptr = length;

} /* end dme */
```

207

```
/****************************************************************
 *
 *   SOURCE FILE:      airspeed.c
 *
 *
 *   FUNCTION:         airspeed()
 *
 *
 *   DESCRIPTION:      This routine retrieves the airspeed
 *                     value from the data package and
 *                     converts it into knots.  It then
 *                     converts this value into a zero-
 *                     padded plot string.
 *
 *
 *   DOCUMENTATION
 *   FILES:            None.
 *
 *
 *   ARGUMENTS:        page_number    : page being displayed
 *                     clock_pkg      : pointer to clock
 *                                      package
 *                     alarm_pkg      : pointer to alarm
 *                                      package
 *                     airspeed_XO    : plot coord.
 *                     airspeed_YO    : plot coord.
 *                     length_ptr     : pointer to length of
 *                                      conversion array
 *                     airspeed_conv  : pointer to conversion
 *                                      array
 *
 *
 *   RETURN:           None.
 *
 *
 *   FUNCTIONS
 *   CALLED:           zero_pad()
 *                     string_gen()
 *                     strcpy()
 *
 *
 *   AUTHOR:           CHUCK ROBERTSON
 *
 *
 *   DATE CREATED:     23Mar87      Version 1.0
 *
 *
 *   REVISIONS:        04Apr87      Version 1.1
 *                                  Added alarm capability.
 *
```

```
 *
*****************************************************************/
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "data_str.h"    /* data package structure
                            declaration.         */

void airspeed(page_number, clock_pkg, alarm_pkg,
              airspeed_XO, airspeed_YO,
              length_ptr, airspeed_conv)

int page_number;
CLOCK_PKG *clock_pkg;
ALARM_PKG *alarm_pkg;
int airspeed_XO, airspeed_YO;
int *length_ptr;
unsigned short airspeed_conv[];

{
    unsigned short string_conv[30];
    int airspeed;
    int string_length;
    int index = 0;
    int i;
    char final_string[20];
    double size1_5 = 1.5;

    void string_gen();
    void zero_pad();

    airspeed = (int)((data_pkg.AIRSPEED + 60) * 0.8427);

    if(airspeed <= 51)
      airspeed = 0;

    /* DISPLAY AIRSPEED ON DATA PAGE AND NAV PAGE */

    if(page_number != 3)
      {
        zero_pad((double)(airspeed),100.0,6,"int",
                 final_string);
        string_gen(final_string, airspeed_XO, airspeed_YO,
                   size1_5, &string_length, string_conv);

        for(i=0;i<string_length;i++)
            airspeed_conv[index + i] = string_conv[i];

        index = index + string_length;
```
209

```c
        }

        /* IF AIRSPEED IS AT STALL SPEED SOUND ALARM */

        if((alarm_pkg->airspeed_alarm_flag == ALARM_OFF)
          && (airspeed < 70))
            {
            alarm_pkg->airspeed_alarm_flag = ALARM_ON;

            if(alarm_pkg->alarm_status_flag == ALARM_OFF)
                {
                TOGGLE_ALARM_SWITCH();
                alarm_pkg->alarm_status_flag = ALARM_ON;
                }
            }

        if((alarm_pkg->airspeed_alarm_flag == ALARM_ON)
          && (airspeed > 75))
            {
            alarm_pkg->airspeed_alarm_flag = ALARM_OFF;

            if((alarm_pkg->alarm_status_flag == ALARM_ON) &&
              (alarm_pkg->assigned_altitude_alarm_flag ==
               ALARM_OFF)
              && (alarm_pkg->mda_dh_alarm_flag == ALARM_OFF)
              && (alarm_pkg->time_out_alarm_flag == ALARM_OFF))
                {
                TOGGLE_ALARM_SWITCH();
                alarm_pkg->alarm_status_flag = ALARM_OFF;
                }
            }

        /* IF ALARM IS ON DISPLAY IMPENDING STALL MESSAGE */

        if(alarm_pkg->airspeed_alarm_flag == ALARM_ON)
            {
                strcpy(final_string, ">>>STALL<<<");
                string_gen(final_string, 1365, 80,
                            size1_5, &string_length, string_conv);

                for(i=0;i<string_length;i++)
                    airspeed_conv[index + i] = string_conv[i];

                index = index + string_length;
            }

    *length_ptr = index;

} /* end airspeed */
```

210

```
/*******************************************************************
 *
 *  SOURCE FILE:     arrow.c
 *
 *
 *  FUNCTION:        arrow()
 *
 *
 *  DESCRIPTION:     This routine generates the HP code to
 *                   put up the head of an arrow for a
 *                   given radius, location, angle, and
 *                   direction. The direction determines
 *                   whether or not the arrow head points
 *                   "to" or "from" the plot coordinates.
 *
 *
 *  DOCUMENTATION
 *  FILES:           None.
 *
 *
 *  ARGUMENTS:       x_center         : x plot coordinate
 *                   y_center         : y plot coordinate
 *                   angle_deg        : plot angle in
 *                                      degrees
 *                   radius           : length of arrow
 *                   dir_string       : direction string
 *                   length_ptr       : pointer to length of
 *                                      conv[] array
 *                   conv             : pointer to
 *                                      conversion array
 *
 *  RETURN:          None.
 *
 *
 *  FUNCTIONS
 *  CALLED:          None.
 *
 *
 *  AUTHOR:          CHUCK ROBERTSON
 *
 *
 *  DATE CREATED:    15Feb87      Version 1.0
 *
 *
 *  REVISIONS:       None.
 *
 *
 *******************************************************************/
#include<math.h>
#include<string.h>
```

211

```c
#include<stdio.h>

void arrow(x_center, y_center, angle_deg, radius, dir_string,
           length_ptr, conv)

unsigned short conv[];
unsigned short *length_ptr;
int x_center, y_center;
double radius;
double angle_deg;
char dir_string[];

{   int      x, y;                    /* plot coordinates */
    int      arrow_head_end_X;        /* arrow head end coord */
    int      arrow_head_end_Y;
    int      arrow_head_bgn_X;        /* arrow head base coord */
    int      arrow_head_bgn_Y;
    int      disp;                    /* plot displacement */
    double PI;
    double angle_rad;                 /* plot angle in radians */
    double screen_scale;

    PI = 3.14159265;
    screen_scale = 9.5/12.5;

    /*  convert angle to radians */

    angle_rad = angle_deg * (2 * PI) / 360.0;

    /* figure displacement for arrow direction */

    if( strcmp(dir_string, "to") == 0 )
       disp = -50;
    else if( strcmp(dir_string, "from") == 0 )
       disp = 50;
    else
       printf("illegal arrow direction inputed\n");

    /* calculate coordinates */

    arrow_head_end_X = (int)(radius * cos(angle_rad)
                                * screen_scale);
    arrow_head_end_Y = (int)(radius * sin(angle_rad));

    conv[0] = 0x0000 | (arrow_head_end_X + x_center);
    conv[1] = 0x1000 | (arrow_head_end_Y + y_center);

    arrow_head_bgn_X = (int)((radius + disp)
                                * cos(angle_rad)
```

212

```
                              * screen_scale);
    arrow_head_bgn_Y = (int)((radius + disp)
                              * sin(angle_rad));

    y = (int)(20 * cos(angle_rad));
    x = (int)(-20 * sin(angle_rad) * screen_scale);

    conv[2] = 0x0000 | (x_center + arrow_head_bgn_X + x);
    conv[3] = 0x1800 | (y_center + arrow_head_bgn_Y + y);

    conv[4] = 0x0000 | (arrow_head_end_X + x_center);
    conv[5] = 0x1000 | (arrow_head_end_Y + y_center);

    conv[6] = 0x0000 | (x_center + arrow_head_bgn_X - x);
    conv[7] = 0x1800 | (y_center + arrow_head_bgn_Y - y);

    *length_ptr = 8;

    ) /* end of arrow */
```

```
/****************************************************************
 *
 *  SOURCE FILE:      heading.c
 *
 *
 *  FUNCTION:         get_heading()
 *
 *
 *  DESCRIPTION:      This routine gets the raw heading from
 *                    the data package and uses a lookup
 *                    table along with interpolation to
 *                    convert the raw heading into the
 *                    actual plane heading.
 *
 *
 *  DOCUMENTATION
 *  FILES:            None.
 *
 *
 *  ARGUMENTS:        plane_heading_deg_ptr : pointer to the
 *                               heading variable in calling
 *                               routine
 *
 *
 *  RETURN:           None.
 *
 *
 *  FUNCTIONS
 *  CALLED:           None.
 *
 *
 *  AUTHOR:           CHUCK ROBERTSON
 *
 *
 *  DATE CREATED:     16Feb87      Version 1.0
 *
 *
 *  REVISIONS:        None.
 *
 *
 ****************************************************************/
#include "data_str.h"
#include<stdlib.h>

typedef struct { double a_to_d;
                 double value;  } CONVERSION;

/* lookup table for heading conversion */

CONVERSION heading[15] = {
```

214

```
                                (0.0, 99.0),
                                (6.0, 90.0),
                                (30.0, 60.0),
                                (50.0, 30.0),
                                (73.0, 0.0),
                                (94.0, 330.0),
                                (115.0, 300.0),
                                (137.0, 270.0),
                                (158.0, 240.0),
                                (180.0, 210.0),
                                (201.0, 180.0),
                                (223.0, 150.0),
                                (245.0, 120.0),
                                (251.0, 110.0),
                                (255.0, 100.0)
                        );

void get_heading(plane_heading_deg_ptr)

int *plane_heading_deg_ptr;

{
    int i;                  /* loop counter */
    double raw_heading;

    /* GET RAW HEADING FROM A/D AND CONVERT
       TO DEGREES USING LOOKUP TABLE        */

    raw_heading = (double)(data_pkg.COMPASS);

    if( (raw_heading > 73.0) && (raw_heading <= 94.0))
        heading[4].value = 360.0;
    else
        heading[4].value = 0.0;

    for(i = 0; i < 14; i++)
        if( (raw_heading >= heading[i].a_to_d) &&
            (raw_heading <= heading[i + 1].a_to_d) )
        {
            /* INTERPOLATE */
            *plane_heading_deg_ptr

                =   (int)
                    ( (raw_heading - heading[i].a_to_d) /
                    (heading[i+1].a_to_d -
                     heading[i].a_to_d) *
                    (heading[i+1].value -
                     heading[i].value) +
                     heading[i].value);
            break;
```

215

```
        }
}   /* end of get_heading */
```

```
/**************************************************************
 *
 *  SOURCE FILE:      timer.c
 *
 *
 *  FUNCTION:         timer()
 *
 *
 *  DESCRIPTION:      This function implements the TIMER and
 *                    TIME_OUT portion of the DATA PAGE.  The
 *                    function keeps track of the timer
 *                    value and sounds an alarm when the
 *                    time-out is reached.
 *
 *
 *  DOCUMENTATION
 *  FILES:            None.
 *
 *
 *  ARGUMENTS:        page_number: page being displayed
 *                    clock_pkg  : pointer to clock package
 *                    alarm_pkg  : pointer to alarm package
 *                    length_ptr : point to length of conv[]
 *                                 array
 *                    timer_conv : pointer to conversion
 *                                 array
 *
 *
 *  RETURN:           None.
 *
 *
 *  FUNCTIONS
 *  CALLED:           string_gen()
 *                    zero_pad()
 *
 *
 *  AUTHOR:           Chuck Robertson
 *
 *
 *  DATE CREATED:     26Feb87      Version 1.0
 *
 *
 *  REVISIONS:        04Apr87      Version 1.1
 *                                 Added alarm capability.
 *
 *
 **************************************************************/
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
```

```c
#include "data_str.h"    /* data package structure
                            declaration.          */
#include "datpg_xy.h"


void timer(page_number,clock_pkg,alarm_pkg,length_ptr,
           timer_conv)

CLOCK_PKG *clock_pkg;
ALARM_PKG *alarm_pkg;
unsigned short timer_conv[];
int *length_ptr;
int page_number;

{
    static int real_time_start_sec;
    static int total_time_out_sec;

    unsigned short string_conv[30];
    int real_time_present_sec;
    int total_timer_present_sec;
    int string_length;
    int i;
    int index = 0;

    double size1_5 = 1.5;

    char final_string[20], cat_string[5];

    void string_gen();
    void zero_pad();


    /* SET TIMER */

    if(clock_pkg->timer_operation_flag == SET_TIMER)
        {
            /* turn off timer */
            clock_pkg->timer_status_flag = TIMER_OFF;

            /* clear operation flag */
            clock_pkg->timer_operation_flag = NULL_TIMER;

            if(alarm_pkg->time_out_alarm_flag == ALARM_ON)
                {
                    alarm_pkg->time_out_alarm_flag = ALARM_OFF;

                    if((alarm_pkg->alarm_status_flag ==
                        ALARM_ON) &&
                       (alarm_pkg->assigned_altitude_alarm_flag
```

218

```
                == ALARM_OFF)
             && (alarm_pkg->mda_dh_alarm_flag ==
                 ALARM_OFF)
             && (alarm_pkg->airspeed_alarm_flag ==
                 ALARM_OFF))
             {
                 TOGGLE_ALARM_SWITCH();
                 alarm_pkg->alarm_status_flag =
                                           ALARM_OFF;
             }
          )
      }

  /* DETERMINE RESET CONDITION */

  if(clock_pkg->timer_operation_flag == RESET_TIMER)
     {
         clock_pkg->timer_min = 0;
         clock_pkg->timer_sec = 0;

         clock_pkg->timer_status_flag = TIMER_OFF;

         clock_pkg->timer_operation_flag = NULL_TIMER;

         if(alarm_pkg->time_out_alarm_flag == ALARM_ON)
            {
                alarm_pkg->time_out_alarm_flag = ALARM_OFF;

                if((alarm_pkg->alarm_status_flag ==
                   ALARM_ON) &&
                   (alarm_pkg->assigned_altitude_alarm_flag
                   == ALARM_OFF)
                 && (alarm_pkg->mda_dh_alarm_flag ==
                     ALARM_OFF)
                 && (alarm_pkg->airspeed_alarm_flag ==
                     ALARM_OFF))
                {
                   TOGGLE_ALARM_SWITCH();
                   alarm_pkg->alarm_status_flag =
                                             ALARM_OFF;
                }
            )
     )

  /* START CLOCK */

  if(clock_pkg->timer_operation_flag == START_TIMER)
     {
         total_time_out_sec = ((60 *
                                clock_pkg->time_out_min) +
```

219

```
                                clock_pkg->time_out_sec);

        real_time_start_sec = (int)
                             ((3600 * data_pkg.HOURS) +
                              (60 * data_pkg.MINUTES) +
                              data_pkg.SECONDS);

        clock_pkg->timer_status_flag = TIMER_ON;

        clock_pkg->timer_operation_flag = NULL_TIMER;

        if(alarm_pkg->time_out_alarm_flag == ALARM_ON)
            {
                alarm_pkg->time_out_alarm_flag = ALARM_OFF;

                if((alarm_pkg->alarm_status_flag ==
                    ALARM_ON) &&
                  (alarm_pkg->assigned_altitude_alarm_flag
                   == ALARM_OFF)
                  && (alarm_pkg->mda_dh_alarm_flag ==
                     ALARM_OFF)
                  && (alarm_pkg->airspeed_alarm_flag ==
                     ALARM_OFF))
                  {
                     TOGGLE_ALARM_SWITCH();
                     alarm_pkg->alarm_status_flag =
                                                ALARM_OFF;
                  }
            }
    }

/* CALCULATE TIMER VALUE AND DISPLAY */

if(clock_pkg->timer_status_flag == TIMER_ON)
    {
        real_time_present_sec = (int)
                               ((3600 * data_pkg.HOURS) +
                                (60 * data_pkg.MINUTES) +
                                data_pkg.SECONDS);

        if(real_time_present_sec < real_time_start_sec)
            real_time_present_sec = real_time_present_sec +
                               (24 * 3600);

        total_timer_present_sec = real_time_present_sec -
                               real_time_start_sec;

        /* convert present timer value into minutes and
           sec
                                                        */
```

```
        clock_pkg->timer_sec = total_timer_present_sec
                               % 60;

        clock_pkg->timer_min = ((total_timer_present_sec -
                                clock_pkg->timer_sec)
                                / 60);

        /* check for time_out alarm */

        if((total_timer_present_sec >= total_time_out_sec)
           && (alarm_pkg->time_out_alarm_flag ==
               ALARM_OFF))
            {
                alarm_pkg->time_out_alarm_flag = ALARM_ON;

                if(alarm_pkg->alarm_status_flag ==
                   ALARM_OFF)
                   {
                   TOGGLE_ALARM_SWITCH();
                   alarm_pkg->alarm_status_flag = ALARM_ON;
                   }
            }

        /* is timer done ? */

        if(total_timer_present_sec == 3600)
           clock_pkg->timer_status_flag = TIMER_OFF;

    }

/* IF ALARM IS ON DISPLAY TIME-OUT MESSAGE */

if(alarm_pkg->time_out_alarm_flag == ALARM_ON)
    {
        strcpy(final_string, ">>TIME-OUT<<");
        string_gen(final_string, 1365, 10,
                   size1_5, &string_length, string_conv);

        for(i=0;i<string_length;i++)
            timer_conv[index + i] = string_conv[i];

        index = index + string_length;
    }

/* DISPLAY TIMER AND TIME-OUT VALUES IF PAGE
   BEING DISPLAYED IS THE DATA PAGE            */

if(page_number == 1)
   {
      /* DISPLAY TIMER VALUE */
```

221

```
            zero_pad((double)(clock_pkg->timer_min),10.0,6,
                    "int",final_string);
            strcat(final_string,":");
            zero_pad((double)(clock_pkg->timer_sec),10.0,6,
                    "int",cat_string);
            strcat(final_string,cat_string);
            string_gen(final_string,timer_X1,timer_Y1,size1_5,
                    &string_length,string_conv);

            for(i=0;i<string_length;i++)
                timer_conv[index + i] = string_conv[i];

            index = index + string_length;

            /* DISPLAY TIME-OUT VALUE */

            zero_pad((double)(clock_pkg->time_out_min),10.0,6,
                    "int",final_string);
            strcat(final_string,":");
            zero_pad((double)(clock_pkg->time_out_sec),10.0,6,
                    "int",cat_string);
            strcat(final_string,cat_string);
            string_gen(final_string,time_out_X1,time_out_Y1,
                    size1_5,&string_length,string_conv);

            for(i=0;i<string_length;i++)
                timer_conv[index + i] = string_conv[i];

            index = index + string_length;
        }

    /* send back length of conversion array */

    *length_ptr = index;

}   /* end of timer */
```

222

APPENDIX D (CONT.)

PAGE-SPECIFIC TOOLS

```
/***************************************************************
 *
 *   SOURCE FILE:      time_gen.c
 *
 *
 *   FUNCTION:         time_string_gen()
 *
 *
 *   DESCRIPTION:      Generates the string depicting the
 *                     time as the values of hours, minutes,
 *                     and seconds of DATA_PKG show. Spaces
 *                     and zeros are inserted in the
 *                     appropriate spots so that the length
 *                     of the final string is always
 *                     constant.
 *
 *
 *   DOCUMENTATION
 *   FILES:            None.
 *
 *
 *   ARGUMENTS:        None.
 *
 *
 *   RETURN:           char *
 *                     pointer to the string
 *                     named time_string.
 *
 *
 *   FUNCTIONS
 *   CALLED:           None.
 *
 *
 *   AUTHOR:           Dave Gruenbacher
 *
 *
 *   DATE CREATED:     07Feb87      Version 1.0
 *
 *
 *   REVISIONS:        None.
 *
 ***************************************************************/
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "data_str.h"    /* data package structure
                            declaration.           */

time_string_gen(time_string)
char time_string[9];
```

224

```
{
    char   temp_string[3];
    char   *strcat(), *itoa();
    int    radix = 10;

    /* start time_string with EOL for strcat() to
        detect.                                   */
    time_string[0] = '\0';

    /* convert data_pkg. HOURS to a character
        string.             */
    itoa(data_pkg. HOURS, temp_string, radix);

    /* if data_pkg. HOURS is less then 10, then insert */
    /* a blank where the tens digit would be.   */
    if (data_pkg. HOURS < 10)
        strcat(time_string, " ");

    /* append hours and a colon to time_string. */
    strcat(time_string, temp_string);
    strcat(time_string, ":");

    /* if data_pkg. MINUTES is less then 10, insert */
    /* a zero where the tens digit would be. */
    if (data_pkg. MINUTES <10)
        strcat(time_string, "O");

    /* convert data_pkg. MINUTES to a character
        string.                                   */
    itoa(data_pkg. MINUTES, temp_string, radix);

    /* append minutes and a colon to time_string. */
    strcat(time_string, temp_string);
    strcat(time_string, ":");

    /* if data_pkg. SECONDS is less then 10, insert a */
    /* zero where the tens digit would be.   */
    if (data_pkg. SECONDS < 10)
        strcat(time_string, "O");

    /* convert data_pkg. SECONDS to a character string. */
    itoa(data_pkg. SECONDS, temp_string, radix);

    /* append seconds and a colon to time_string. */
    strcat(time_string, temp_string);

    /* return pointer to time_string to calling routine. */
    return;

} /* time_string_gen */
```

```
/*****************************************************************
 *
 *  SOURCE FILE:      clim_box.c
 *
 *
 *  FUNCTION:         clim_box()
 *
 *
 *  DESCRIPTION:      This program generates the HP code to
 *                    put up the climb indicator box for the
 *                    DATA PAGE on the vector graphics
 *                    display.
 *
 *  DOCUMENTATION
 *  FILES:            None.
 *
 *
 *  ARGUMENTS:        XO, YO       : plot coordinates for the
 *                                   box
 *                    length_ptr   : pointer to the length of
 *                                   conv[] array
 *                    conv         : pointer to converion
 *                                   array
 *
 *  RETURN:           None.
 *
 *
 *  FUNCTIONS
 *  CALLED:           None.
 *
 *
 *  AUTHOR:           CHUCK ROBERTSON
 *
 *
 *  DATE CREATED:     27Jan87      Version 1.0
 *
 *
 *  REVISIONS:        None.
 *
 *
 *****************************************************************/

void climb_box(XO, YO, length_ptr, conv)

unsigned short XO, YO;
unsigned short *length_ptr;
unsigned short conv[];

{
   int i;                       /* loop counter */
```

226

```c
    int X, Y;                        /* plot coordinate variables */

    /* move to initial coordinates and generate
       box coordinates                           */

    X = X0;
    Y = Y0;

    conv[0] = 0x0000 | X;   /* plot x off */
    conv[1] = 0x1000 | Y;   /* plot y off */

    X = X - 60;
    Y = Y + 315;
    conv[2] = 0x0000 | X;
    conv[3] = 0x1800 | Y;

    X = X - 80;
    conv[4] = 0x0000 | X;
    conv[5] = 0x1800 | Y;

    Y = Y - 630;
    conv[6] = 0x0000 | X;
    conv[7] = 0x1800 | Y;

    X = X + 80;
    conv[8] = 0x0000 | X;
    conv[9] = 0x1800 | Y;

    X = X + 60;
    Y = Y + 315;
    conv[10] = 0x0000 | X;
    conv[11] = 0x1800 | Y;

    *length_ptr = 12;

}   /* end of clim_box */
```

227

```
/****************************************************************
 *
 *    SOURCE FILE:        clim_hsh.c
 *
 *
 *    FUNCTION:           clim_hsh()
 *
 *
 *    DESCRIPTION:        This routine generates the HP code to
 *                        put up the hash marks of the climb
 *                        indicator of the DATA PAGE.
 *
 *
 *    DOCUMENTATION
 *    FILES:              None.
 *
 *
 *    ARGUMENTS:          x_center          : x plot coordinate
 *                                            for hash arc center
 *                        y_center          : y plot coordinate
 *                                            for hash arc center
 *                        start_angle_deg : initial hash angle
 *                        end_angle_deg   : final hash angle
 *                        radius          : radius of hash arc
 *                        length_ptr      : pointer to length of
 *                                          conv[] array
 *                        conv            : pointer to
 *                                          conversion array
 *
 *    RETURN:             None.
 *
 *
 *    FUNCTIONS
 *    CALLED:             None.
 *
 *
 *    AUTHOR:             CHUCK ROBERTSON
 *
 *
 *    DATE CREATED:       25Jan87      Version 1.0
 *
 *
 *    REVISIONS:          None.
 *
 *
 ****************************************************************/
#include<math.h>

void clim_hsh(x_center,y_center,start_angle_deg,
              end_angle_deg,radius,length_ptr,conv)
```

228

```
unsigned short conv[];
unsigned short *length_ptr;
int x_center,y_center;
int radius;
double start_angle_deg;
double end_angle_deg;

{
    int     X,Y;                    /* plot coordinate
                                       variables        */
    int     final_delta_cnt;        /* iteration number */
    int     delta_count;            /* loop counter */
    int     i;                      /* loop counter */
    int     index;                  /* conversion index */

    double PI;
    double DEGREE_TO_RAD;
    double start_angle_rad;         /* beginning angle in
                                       radians          */
    double end_angle_rad;           /* ending angle in
                                       radians          */
    double delta_theta_deg;         /* subinterval of total
                                       angle in degrees    */
    double delta_theta_rad;         /* subinterval of total
                                       angle in radians    */
    double theta;                   /* calculation angle */
    double total_angle;             /* total arc angle */
    double screen_scale;            /* screen scale for X
                                       direction           */

    PI = 3.14159265;
    screen_scale = 9.5/12.5;
    DEGREE_TO_RAD = (2 * PI) / 360.0;
    index = 0;

    /* convert beginning and ending angles to radians */

    start_angle_rad = start_angle_deg * DEGREE_TO_RAD;
    end_angle_rad = end_angle_deg * DEGREE_TO_RAD;

    /* initialize calculation angle */

    theta = start_angle_rad;

    /* calculate total arc angle */

    total_angle = end_angle_deg - start_angle_deg;

    /* calculate subinterval */
```

229

```c
delta_theta_deg = total_angle/4.0;
delta_theta_rad = delta_theta_deg * DEGREE_TO_RAD;

final_delta_cnt = 4;   /* iteration # */

/* calculate coordinates */

for(delta_count = 0; delta_count <= final_delta_cnt;
    delta_count++)
   {
      if(delta_count != 2)
        {
          X = (int)((radius - 23) * cos(theta) *
                     screen_scale);
          Y = (int)((radius - 23) * sin(theta));

          conv[index++] = 0x0000 | (X + x_center);
          conv[index++] = 0x1000 | (Y + y_center);

          X = (int)((radius + 23) * cos(theta) *
                     screen_scale);
          Y = (int)((radius + 23) * sin(theta));

          conv[index++] = 0x0000 | (X + x_center);
          conv[index++] = 0x1800 | (Y + y_center);
        }

      theta = theta + delta_theta_rad;

   }

*length_ptr = index;

) /* end of clim_hsh */
```

```
/**************************************************************
 *
 *   SOURCE FILE:      clim_aro.c
 *
 *
 *   FUNCTION:         clim_arrow()
 *
 *
 *   DESCRIPTION:      This routine generates the HP code to
 *                     put up the climb arrow for the rate of
 *                     climb indicator on the DATA PAGE. It
 *                     makes the  calculation for a given
 *                     location, radius, and angle.
 *
 *
 *   DOCUMENTATION
 *   FILES:            None.
 *
 *
 *   ARGUMENTS:        x_center        : x plot coordinate
 *                                       for arrow
 *                     y_center        : y plot coordinate
 *                                       for arrow
 *                     angle_deg       : plot angle in
 *                                       degrees
 *                     radius          : length of arrow
 *                     length_ptr      : pointer to length of
 *                                       conv[] array
 *                     conv            : pointer to
 *                                       conversion array
 *
 *   RETURN:           None.
 *
 *
 *   FUNCTIONS
 *   CALLED:           None.
 *
 *
 *   AUTHOR:           CHUCK ROBERTSON
 *
 *
 *   DATE CREATED:     02Feb87     Version 1.0
 *
 *
 *   REVISIONS:        None.
 *
 *
 **************************************************************/
#include<math.h>
```

231

```c
void clim_arrow(x_center,y_center,angle_deg,radius,
                length_ptr,conv)

unsigned short conv[];
unsigned short *length_ptr;
int x_center,y_center;
int radius;
double angle_deg;

(   int     x,y;                      /* plot coordinates */
    int     arrow_head_end_X;         /* arrow head end coord */
    int     arrow_head_end_Y;
    int     arrow_head_bgn_X;         /* arrow head base coord */
    int     arrow_head_bgn_Y;
    double  PI;
    double  angle_rad;                /* plot angle in radians */
    double  screen_scale;

    PI = 3.14159265;
    screen_scale = 9.5/12.5;

    /*  convert angle to radians */

    angle_rad = angle_deg * (2 * PI) / 360.0;

    /* calculate coordinates */

    conv[0] = 0x0000 | x_center;    /* plot x off */
    conv[1] = 0x1000 | y_center;    /* plot y off */

    arrow_head_end_X = (int)(radius * cos(angle_rad)
                             * screen_scale);
    arrow_head_end_Y = (int)(radius * sin(angle_rad));

    conv[2] = 0x0000 | (arrow_head_end_X + x_center);
    conv[3] = 0x1800 | (arrow_head_end_Y + y_center);

    arrow_head_bgn_X = (int)((radius - 50) *
                             cos(angle_rad) *
                             screen_scale);
    arrow_head_bgn_Y = (int)((radius - 50) *
                             sin(angle_rad));

    y = (int)(20 * cos(angle_rad));
    x = (int)(-20 * sin(angle_rad) * screen_scale);

    conv[4] = 0x0000 | (x_center + arrow_head_bgn_X + x);
    conv[5] = 0x1800 | (y_center + arrow_head_bgn_Y + y);

    conv[6] = 0x0000 | (arrow_head_end_X + x_center);
```

232

```c
conv[7] = 0x1000 | (arrow_head_end_Y + y_center);

conv[8] = 0x0000 | (x_center + arrow_head_bgn_X - x);
conv[9] = 0x1800 | (y_center + arrow_head_bgn_Y - y);

*length_ptr = 10;

} /* end of clim_arrow */
```

```
/*****************************************************************
 *
 *   SOURCE FILE:     climrate.c
 *
 *
 *   FUNCTION:        climb_rate()
 *
 *
 *   DESCRIPTION:     This routine retrieves the vertical
 *                    speed value from the data package and
 *                    converts it into feet/minute.   It
 *                    then converts this value into a zero-
 *                    padded plot string.
 *
 *
 *   DOCUMENTATION
 *   FILES:           None.
 *
 *
 *   ARGUMENTS:       clock_pkg        : pointer to clock
 *                                       package
 *                    alarm_pkg        : pointer to alarm
 *                                       package
 *                    XO,YO            : plot coordinates
 *                    vert_speed       : pointer to
 *                                       climb_rate
 *                    length_ptr       : pointer to length of
 *                                       conversion array
 *                    climb_rate_conv  : pointer to
 *                                       conversion array
 *
 *
 *   RETURN:          None.
 *
 *
 *   FUNCTIONS
 *   CALLED:          zero_pad()
 *                    string_gen()
 *
 *
 *   AUTHOR:          CHUCK ROBERTSON
 *
 *
 *   DATE CREATED:    23Mar87      Version 1.0
 *
 *
 *   REVISIONS:       11Apr87
 *                    Added call to climb_filter()
 *                    Dave Gruenbacher
 *
```

234

```c
 *
 ****************************************************************/
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "data_str.h"    /* data package structure
                             declaration.              */


void climb_rate(clock_pkg, alarm_pkg, XO, YO, vert_speed,
                length_ptr, climb_rate_conv)

CLOCK_PKG *clock_pkg;
ALARM_PKG *alarm_pkg;
int XO, YO;
int *length_ptr;
int *vert_speed;
unsigned short climb_rate_conv[];

{
    int climb_rate;
    int length;
    int radix = 10;
    char final_string[20];
    double size = 1.5;
    float climb_filter();

    void string_gen();

    if(data_pkg.VERT_SPEED <= 115)
       data_pkg.VERT_SPEED = data_pkg.VERT_SPEED + 16;

    data_pkg.VERT_SPEED = (unsigned char)
                            (climb_filter(data_pkg.VERT_SPEED));

    climb_rate = (int)((data_pkg.VERT_SPEED - 136.0)
                            * 20.3);

    itoa(climb_rate, final_string, radix);
    string_gen(final_string, XO, YO, size, &length,
               climb_rate_conv);

    *vert_speed = climb_rate;
    *length_ptr = length;

} /* end climb_rate */
```

235

```
/*****************************************************************
 *
 *   SOURCE FILE:      plane.c
 *
 *
 *   FUNCTION:         plane()
 *
 *
 *   DESCRIPTION:      This function generates the code to
 *                     put up a plane on the vector graphics
 *                     display at a given location.
 *
 *
 *   DOCUMENTATION
 *   FILES:            None.
 *
 *
 *   ARGUMENTS:        XO, YO        : plot coordinates for
 *                                     plane
 *                     length_ptr   : pointer to length of
 *                                     conv[] array
 *                     conv         : pointer to conversion
 *                                     array
 *
 *
 *   RETURN:           None.
 *
 *
 *   FUNCTIONS
 *   CALLED:           None.
 *
 *
 *   AUTHOR:           CHUCK ROBERTSON
 *
 *
 *   DATE CREATED:     15Jan87      Version 1.0
 *
 *
 *   REVISIONS:        None.
 *
 *
 *****************************************************************/

void plane(XO, YO, length_ptr, conv)

unsigned short conv[];
unsigned short *length_ptr;
unsigned short XO, YO;

{   int X, Y;               /* coordinate variables */
```

236

```
/* draw fusilage */
    X = X0;
    Y = Y0 - 140;
    conv[0] = 0x0000 | X;
    conv[1] = 0x1000 | Y;

    X = X - 20;
    Y = Y + 180;
    conv[2] = 0x0000 | X;
    conv[3] = 0x1800 | Y;

    X = X + 20;
    Y = Y + 20;
    conv[4] = 0x0000 | X;
    conv[5] = 0x1800 | Y;

    X = X + 20;
    Y = Y - 20;
    conv[6] = 0x0000 | X;
    conv[7] = 0x1800 | Y;

    X = X - 20;
    Y = Y - 180;
    conv[8] = 0x0000 | X;
    conv[9] = 0x1800 | Y;

/* draw wing */
    X = X0 - 100;
    Y = Y0 - 20;
    conv[10] = 0x0000 | X;
    conv[11] = 0x1000 | Y;

    Y = Y + 40;
    conv[12] = 0x0000 | X;
    conv[13] = 0x1800 | Y;

    X = X + 200;
    conv[14] = 0x0000 | X;
    conv[15] = 0x1800 | Y;

    Y = Y - 40;
    conv[16] = 0x0000 | X;
    conv[17] = 0x1800 | Y;

    X = X - 200;
    conv[18] = 0x0000 | X;
    conv[19] = 0x1800 | Y;

/* draw aeleron */
    X = X0 - 30;
```

```
   Y = Y0 - 120;
   conv[20] = 0x0000 | X;
   conv[21] = 0x1000 | Y;

   Y = Y + 20;
   conv[22] = 0x0000 | X;
   conv[23] = 0x1800 | Y;

   X = X + 60;
   conv[24] = 0x0000 | X;
   conv[25] = 0x1800 | Y;

   Y = Y - 20;
   conv[26] = 0x0000 | X;
   conv[27] = 0x1800 | Y;

   X = X - 60;
   conv[28] = 0x0000 | X;
   conv[29] = 0x1800 | Y;

   *length_ptr = 30;

}/* end plane */
```

```
/******************************************************************
 *
 *  SOURCE FILE:        waypoint.c
 *
 *
 *  FUNCTION:           waypoint()
 *
 *
 *  DESCRIPTION:        This function generates the code to
 *                      put up a waypoint on the vector
 *                      graphics display at a given location.
 *
 *
 *  DOCUMENTATION
 *  FILES:              None.
 *
 *
 *  ARGUMENTS:          XO, YO        : plot coordinates for
 *                                      plane
 *                      length_ptr    : pointer to length of
 *                                      conv[] array
 *                      conv          : pointer to conversion
 *                                      array
 *
 *
 *  RETURN:             None.
 *
 *
 *  FUNCTIONS
 *  CALLED:             None.
 *
 *
 *  AUTHOR:             CHUCK ROBERTSON
 *
 *
 *  DATE CREATED:       25Jan87       Version 1.0
 *
 *
 *  REVISIONS:          None.
 *
 *
 ******************************************************************/

void waypoint(XO, YO, length_ptr, conv)

unsigned short conv[];
unsigned short *length_ptr;
unsigned short XO, YO;

{   unsigned short X, Y;                /* coordinate variables */
```

```c
double screen_scale = 9.5/12.5;    /* scale screen */

X = X0 - (unsigned short)(13 * screen_scale);
Y = Y0 + 13;
conv[0] = 0x0000 | X;
conv[1] = 0x1000 | Y;

X = X + (unsigned short)(13 * screen_scale);
Y = Y + 37;
conv[2] = 0x0000 | X;
conv[3] = 0x1800 | Y;

X = X + (unsigned short)(13 * screen_scale);
Y = Y - 37;
conv[4] = 0x0000 | X;
conv[5] = 0x1800 | Y;

X = X + (unsigned short)(38 * screen_scale);
conv[6] = 0x0000 | X;
conv[7] = 0x1800 | Y;

X = X - (unsigned short)(27 * screen_scale);
Y = Y - 19;
conv[8] = 0x0000 | X;
conv[9] = 0x1800 | Y;

X = X + (unsigned short)(14 * screen_scale);
Y = Y - 40;
conv[10] = 0x0000 | X;
conv[11] = 0x1800 | Y;

X = X - (unsigned short)(38 * screen_scale);
Y = Y + 27;
conv[12] = 0x0000 | X;
conv[13] = 0x1800 | Y;

X = X - (unsigned short)(38 * screen_scale);
Y = Y - 27;
conv[14] = 0x0000 | X;
conv[15] = 0x1800 | Y;

X = X + (unsigned short)(14 * screen_scale);
Y = Y + 40;
conv[16] = 0x0000 | X;
conv[17] = 0x1800 | Y;

X = X - (unsigned short)(27 * screen_scale);
Y = Y + 19;
conv[18] = 0x0000 | X;
conv[19] = 0x1800 | Y;
```

```
    X = X + (unsigned short)(38 * screen_scale);
    conv[20] = 0x0000 | X;
    conv[21] = 0x1800 | Y;

    *length_ptr = 22;

} /* end waypoint */
```

```
/***************************************************************
 *
 *   SOURCE FILE:     vortac.c
 *
 *
 *   FUNCTION:        vortac()
 *
 *
 *   DESCRIPTION:     This function generates the code to
 *                    put up a vortac on the vector
 *                    graphics display at a given location.
 *
 *
 *   DOCUMENTATION
 *   FILES:           None.
 *
 *
 *   ARGUMENTS:       XO, YO       : plot coordinates for
 *                                   plane
 *                    length_ptr   : pointer to length of
 *                                   conv[] array
 *                    conv         : pointer to conversion
 *                                   array
 *
 *
 *   RETURN:          None.
 *
 *
 *   FUNCTIONS
 *   CALLED:          None.
 *
 *
 *   AUTHOR:          CHUCK ROBERTSON
 *
 *
 *   DATE CREATED:    25Jan87      Version 1.0
 *
 *
 *   REVISIONS:       None.
 *
 *
 ***************************************************************/

void vortac(XO, YO, length_ptr, conv)

unsigned short conv[];
unsigned short *length_ptr;
unsigned short XO, YO;

{   unsigned short X, Y;                /* coordinates variables */
```

242

```c
        double screen_scale = 9.5/12.5;   /* scale screen */

        X = X0 - (unsigned short)(25 * screen_scale);
        Y = Y0 + 40;
        conv[0] = 0x0000 | X;
        conv[1] = 0x1000 | Y;

        X = X + (unsigned short)(50 * screen_scale);
        conv[2] = 0x0000 | X;
        conv[3] = 0x1800 | Y;

        X = X + (unsigned short)(30 * screen_scale);
        Y = Y - 40;
        conv[4] = 0x0000 | X;
        conv[5] = 0x1800 | Y;

        X = X - (unsigned short)(30 * screen_scale);
        Y = Y - 40;
        conv[6] = 0x0000 | X;
        conv[7] = 0x1800 | Y;

        X = X - (unsigned short)(50 * screen_scale);
        conv[8] = 0x0000 | X;
        conv[9] = 0x1800 | Y;

        X = X - (unsigned short)(30 * screen_scale);
        Y = Y + 40;
        conv[10] = 0x0000 | X;
        conv[11] = 0x1800 | Y;

        X = X + (unsigned short)(30 * screen_scale);
        Y = Y + 40;
        conv[12] = 0x0000 | X;
        conv[13] = 0x1800 | Y;

        Y = Y + 30;
        conv[14] = 0x0000 | X;
        conv[15] = 0x1800 | Y;

        X = X + (unsigned short)(50 * screen_scale);
        conv[16] = 0x0000 | X;
        conv[17] = 0x1800 | Y;

        Y = Y - 30;
        conv[18] = 0x0000 | X;
        conv[19] = 0x1800 | Y;
/* fill in */
        Y = Y + 6;
        conv[20] = 0x0000 | X;
```

```
    conv[21] = 0x1000 | Y;

    X = X - (unsigned short)(50 * screen_scale);
    conv[22] = 0x0000 | X;
    conv[23] = 0x1800 | Y;

    Y = Y + 6;
    conv[24] = 0x0000 | X;
    conv[25] = 0x1000 | Y;

    X = X + (unsigned short)(50 * screen_scale);
    conv[26] = 0x0000 | X;
    conv[27] = 0x1800 | Y;

    Y = Y + 6;
    conv[28] = 0x0000 | X;
    conv[29] = 0x1000 | Y;

    X = X - (unsigned short)(50 * screen_scale);
    conv[30] = 0x0000 | X;
    conv[31] = 0x1800 | Y;

    Y = Y + 6;
    conv[32] = 0x0000 | X;
    conv[33] = 0x1000 | Y;

    X = X + (unsigned short)(50 * screen_scale);
    conv[34] = 0x0000 | X;
    conv[35] = 0x1800 | Y;

    Y = Y + 6;
    conv[36] = 0x0000 | X;
    conv[37] = 0x1000 | Y;

    X = X - (unsigned short)(50 * screen_scale);
    conv[38] = 0x0000 | X;
    conv[39] = 0x1800 | Y;
/* bottom right knob */
    X = X0 + (unsigned short)(25 * screen_scale);
    Y = Y0 - 40;
    conv[40] = 0x0000 | X;
    conv[41] = 0x1000 | Y;

    X = X + (unsigned short)(24 * screen_scale);
    Y = Y - 18;
    conv[42] = 0x0000 | X;
    conv[43] = 0x1800 | Y;

    X = X + (unsigned short)(30 * screen_scale);
```

```
    Y = Y + 40;
    conv[44] = 0x0000 | X;
    conv[45] = 0x1800 | Y;

    X = X - (unsigned short)(24 * screen_scale);
    Y = Y + 18;
    conv[46] = 0x0000 | X;
    conv[47] = 0x1800 | Y;

/* fill in */
    X = X + (unsigned short)(4 * screen_scale);
    Y = Y - 3;
    conv[48] = 0x0000 | X;
    conv[49] = 0x1000 | Y;

    X = X - (unsigned short)(30 * screen_scale);
    Y = Y - 40;
    conv[50] = 0x0000 | X;
    conv[51] = 0x1800 | Y;

    X = X + (unsigned short)(4 * screen_scale);
    Y = Y - 3;
    conv[52] = 0x0000 | X;
    conv[53] = 0x1000 | Y;

    X = X + (unsigned short)(30 * screen_scale);
    Y = Y + 40;
    conv[54] = 0x0000 | X;
    conv[55] = 0x1800 | Y;

    X = X + (unsigned short)(4 * screen_scale);
    Y = Y - 3;
    conv[56] = 0x0000 | X;
    conv[57] = 0x1000 | Y;

    X = X - (unsigned short)(30 * screen_scale);
    Y = Y - 40;
    conv[58] = 0x0000 | X;
    conv[59] = 0x1800 | Y;

    X = X + (unsigned short)(4 * screen_scale);
    Y = Y - 3;
    conv[60] = 0x0000 | X;
    conv[61] = 0x1000 | Y;

    X = X + (unsigned short)(30 * screen_scale);
    Y = Y + 40;
    conv[62] = 0x0000 | X;
    conv[63] = 0x1800 | Y;
```

```
    X = X + (unsigned short)(4 * screen_scale);
    Y = Y - 3;
    conv[64] = 0x0000 | X;
    conv[65] = 0x1000 | Y;

    X = X - (unsigned short)(30 * screen_scale);
    Y = Y - 40;
    conv[66] = 0x0000 | X;
    conv[67] = 0x1800 | Y;

    X = X + (unsigned short)(4 * screen_scale);
    Y = Y - 3;
    conv[68] = 0x0000 | X;
    conv[69] = 0x1000 | Y;

    X = X + (unsigned short)(30 * screen_scale);
    Y = Y + 40;
    conv[70] = 0x0000 | X;
    conv[71] = 0x1800 | Y;
/* bottom left knob */
    X = X0 - (unsigned short)(55 * screen_scale);
    Y = Y0;
    conv[72] = 0x0000 | X;
    conv[73] = 0x1000 | Y;

    X = X - (unsigned short)(24 * screen_scale);
    Y = Y - 18;
    conv[74] = 0x0000 | X;
    conv[75] = 0x1800 | Y;

    X = X + (unsigned short)(30 * screen_scale);
    Y = Y - 40;
    conv[76] = 0x0000 | X;
    conv[77] = 0x1800 | Y;

    X = X + (unsigned short)(24 * screen_scale);
    Y = Y + 18;
    conv[78] = 0x0000 | X;
    conv[79] = 0x1800 | Y;
/* fill in */
    X = X - (unsigned short)(4 * screen_scale);
    Y = Y - 3;
    conv[80] = 0x0000 | X;
    conv[81] = 0x1000 | Y;

    X = X - (unsigned short)(30 * screen_scale);
    Y = Y + 40;
    conv[82] = 0x0000 | X;
```

```
conv[83] = 0x1800 | Y;

X = X - (unsigned short)(4 * screen_scale);
Y = Y - 3;
conv[84] = 0x0000 | X;
conv[85] = 0x1000 | Y;

X = X + (unsigned short)(30 * screen_scale);
Y = Y - 40;
conv[86] = 0x0000 | X;
conv[87] = 0x1800 | Y;

X = X - (unsigned short)(4 * screen_scale);
Y = Y - 3;
conv[88] = 0x0000 | X;
conv[89] = 0x1000 | Y;

X = X - (unsigned short)(30 * screen_scale);
Y = Y + 40;
conv[90] = 0x0000 | X;
conv[91] = 0x1800 | Y;

X = X - (unsigned short)(4 * screen_scale);
Y = Y - 3;
conv[92] = 0x0000 | X;
conv[93] = 0x1000 | Y;

X = X + (unsigned short)(30 * screen_scale);
Y = Y - 40;
conv[94] = 0x0000 | X;
conv[95] = 0x1800 | Y;

X = X - (unsigned short)(4 * screen_scale);
Y = Y - 3;
conv[96] = 0x0000 | X;
conv[97] = 0x1000 | Y;

X = X - (unsigned short)(30 * screen_scale);
Y = Y + 40;
conv[98] = 0x0000 | X;
conv[99] = 0x1800 | Y;

X = X - (unsigned short)(4 * screen_scale);
Y = Y - 3;
conv[100] = 0x0000 | X;
conv[101] = 0x1000 | Y;

X = X + (unsigned short)(30 * screen_scale);
Y = Y - 40;
conv[102] = 0x0000 | X;
```

247

```
    conv[103] = 0x1800 | Y;

    *length_ptr = 104;

}/* end vortac */
```

```
/**************************************************************
 *
 *   SOURCE FILE:    ndb. c
 *
 *
 *   FUNCTION:       ndb()
 *
 *
 *   DESCRIPTION:    This routine generates the HP code to
 *                   put up a ndb at a given location on
 *                   the vector graphics display.
 *
 *
 *   DOCUMENTATION
 *   FILES:          None.
 *
 *
 *   ARGUMENTS:      XO, YO          : plot coordinates
 *                   length_ptr      : pointer to length of
 *                                     conv[] array
 *                   conv            : pointer to
 *                                     conversion array
 *
 *   RETURN:         None.
 *
 *
 *   FUNCTIONS
 *   CALLED:         arc_circ()
 *
 *
 *   AUTHOR:         CHUCK ROBERTSON
 *
 *
 *   DATE CREATED:   30Jan87      Version 1.0
 *
 *
 *   REVISIONS:      None.
 *
 *
 **************************************************************/

void ndb(XO, YO, length_ptr, conv)

unsigned short conv[];
unsigned short *length_ptr;
int XO, YO;

{   unsigned short  i;                  /* loop counter */
    unsigned short  arc_circ_conv[100]; /* conversion
                                           array       */
```

```c
    unsigned short  arc_circ_length;      /* insertion
                                             length    */
    double start_ang, end_ang;
    int    radius1, radius2, radius3;
    int    total_length;                  /* total insertion
                                             length          */
    void arc_circ();

    total_length = 0;
    start_ang = 0.0;
    end_ang = 360.0;
    radius1 = 25;
    radius2 = 50;
    radius3 = 75;

    arc_circ(X0, Y0, start_ang, end_ang, radius1,
             &arc_circ_length, arc_circ_conv);
    for(i=0;i<arc_circ_length;i++)
       conv[total_length + i] = arc_circ_conv[i];
    total_length = total_length + arc_circ_length;

    arc_circ(X0, Y0, start_ang, end_ang, radius2,
             &arc_circ_length, arc_circ_conv);
    for(i=0;i<arc_circ_length;i++)
       conv[total_length + i] = arc_circ_conv[i];
    total_length = total_length + arc_circ_length;

    arc_circ(X0, Y0, start_ang, end_ang, radius3,
             &arc_circ_length, arc_circ_conv);
    for(i=0;i<arc_circ_length;i++)
       conv[total_length + i] = arc_circ_conv[i];
    total_length = total_length + arc_circ_length;

    *length_ptr = total_length;

} /* end of ndb */
```

```
/***************************************************************
 *
 *  SOURCE FILE:      compass.c
 *
 *
 *  FUNCTION:         compass()
 *
 *
 *  DESCRIPTION:      This routine generates the HP code to
 *                    put up the hash marks and appropriate
 *                    markings for a compass showing 90
 *                    degrees at a given radius for a given
 *                    heading.
 *
 *
 *  DOCUMENTATION
 *  FILES:            None.
 *
 *
 *  ARGUMENTS:        x_center        : x plot coordinate
 *                                      for compass center
 *                    y_center        : y plot coordinate
 *                                      for compass center
 *                    heading_deg     : heading in degrees
 *                    radius          : radius of compass
 *                    length_ptr      : pointer to length of
 *                                      conv[] array
 *                    compass_conv    : pointer to
 *                                      conversion array
 *
 *  RETURN:           None.
 *
 *
 *  FUNCTIONS
 *  CALLED:           None.
 *
 *
 *  AUTHOR:           CHUCK ROBERTSON
 *
 *
 *  DATE CREATED:     04Feb87      Version 1.0
 *
 *
 *  REVISIONS:        None.
 *
 *
 ***************************************************************/
#include<math.h>
#include<string.h>
#include<stdlib.h>
```

```
void compass(x_center, y_center, heading_deg, radius,
             length_ptr, compass_conv)

unsigned short compass_conv[];
unsigned short *length_ptr;
int x_center, y_center;
int radius;
int heading_deg;

{   int     X, Y;                       /* plot coordinates */
    int     i;                          /* loop counter */
    int     length;                     /* insertion length */
    int     heading_upper_bound;        /* heading bounds */
    int     heading_lower_bound;
    int     degrees_off_center;         /* degrees off center
                                           from nearest tick
                                           mark              */
    int     nearest_tick;               /* nearest tick to
                                           heading           */
    int     upper_tick, lower_tick;     /* max and min tick
                                           values            */
    int     tick_value;                 /* value of tick mark */
    int     num_iterations;             /* num of loop
                                           iterations   */
    int     count;                      /* loop counter */
    int     index;                      /* conversion array
                                           index             */
    int     disp;                       /* displacement for
                                           display           */
    int     str_length;                 /* string insertion
                                           length            */
    int     tick_value_X0;              /* tick value plot
                                           coord             */
    int     tick_value_Y0;

    double PI;
    double delta_theta_deg;             /* angle increment
                                           degrees           */
    double delta_theta_rad;             /* angle increment
                                           radians           */
    double theta_rad, theta_deg;        /* calculation angle */
    double screen_scale;
    double DEGREE_TO_RAD;
    double start_angle_rad;             /* first calculation
                                           angle             */
    double start_angle_deg;
    double size;                        /* character size */

    char    hdg_string[3];
```

252

```
unsigned short string_conv[40];

void string_gen();

PI = 3.14159265;
delta_theta_deg = 5.0;
DEGREE_TO_RAD = (2 * PI / 360.0);
screen_scale = 9.5/12.5;
size = 1.0;
length = 0;

/* calculate upper and lower bounds of heading */

heading_upper_bound = (heading_deg + 45) % 360;
heading_lower_bound = (heading_deg + 315) % 360;

/* calculate degrees off center of closest tick mark
   from heading                                      */

degrees_off_center = heading_deg % 5;

/* calculate what tick is nearest heading */

nearest_tick = heading_deg - degrees_off_center;

/* calculate upper and lower ticks */

upper_tick = (nearest_tick + 45) % 360;
lower_tick = (nearest_tick + 315) % 360;

/* get lower_tick within range */

if(lower_tick < heading_lower_bound)
   lower_tick = (lower_tick + 5) % 360;

/* calculate start plot angle for lower_tick */

if(lower_tick < heading_lower_bound)
   start_angle_deg = (double)(135.0 -
                     ((lower_tick + 360) -
                     heading_lower_bound));
else
   start_angle_deg = (double)(135.0 -
                     (lower_tick -
                      heading_lower_bound));

/* convert angles to radians */

delta_theta_rad = delta_theta_deg * DEGREE_TO_RAD;
start_angle_rad = start_angle_deg * DEGREE_TO_RAD;
```

```c
/* INITIALIZE CALCULATION ANGLE, TICK_VALUE, AND
   INDEX                                          */

theta_rad = start_angle_rad;
tick_value = lower_tick;
index = 0;

/* CALCULATE # OF ITERATIONS */

if((heading_upper_bound % 5) == 0)
   num_iterations = 19;
else
   num_iterations = 18;

/* CALCULATE COORDINATES AND CHARACTERS FOR HP CODE */

for(count = 0; count < num_iterations; count++)
   {
       X = (int)(radius * cos(theta_rad) * screen_scale);
       Y = (int)(radius * sin(theta_rad));

       compass_conv[index++] = 0x0000 | (X + x_center);
       compass_conv[index++] = 0x1000 | (Y + y_center);

       if((tick_value % 10) == 0)
          disp = 100;
       else
          disp = 50;

       X = (int)((radius - disp) * cos(theta_rad)
                  * screen_scale);
       Y = (int)((radius - disp) * sin(theta_rad));

       compass_conv[index++] = 0x0000 | (X + x_center);
       compass_conv[index++] = 0x1800 | (Y + y_center);

       theta_rad = theta_rad - delta_theta_rad;

       if((tick_value % 90) == 0)
          {
              switch (tick_value)
                {
                    case 0:
                      strcpy(hdg_string,"N");
                      break;
                    case 90:
                      strcpy(hdg_string,"E");
                      break;
                    case 180:
                      strcpy(hdg_string,"S");
```

254

```
                    break;
                case 270:
                    strcpy(hdg_string, "W");
                    break;
                default:
                    break;
            }
            str_length = 1;
        }
    else   if((tick_value % 10) == 0)
              {
                  itoa(tick_value,hdg_string,10);
                  str_length = strlen(hdg_string);
              }

    /* CALCULATE COORDINATES FOR TICK_VALUE */

    switch (str_length)
        {
            case 1:
                disp = 18;
                break;
            case 2:
                disp = 36;
                break;
            case 3:
                disp = 54;
                break;
        }

    tick_value_XO = X + x_center - disp;
    tick_value_YO = Y + y_center - 50;

    if((tick_value % 10) == 0)
        {
            string_gen(hdg_string,tick_value_XO,
                        tick_value_YO,size,&length,
                        string_conv);

            for(i=0;i<length;i++)
                compass_conv[index + i] = string_conv[i];

            index = index + length;
        }

    tick_value = (tick_value + 5) % 360;

    }

*length_ptr = index;
```

255

```
}/* end of compass */
```

```
/************************************************************
 *
 *   SOURCE FILE:      hdg_brg.c
 *
 *
 *   FUNCTION:         heading_and_bearing()
 *
 *
 *   DESCRIPTION:      This routine generates the HP code to
 *                     put up the heading and bearing of an
 *                     ndb or vortac.  It calculates the
 *                     appropriate value and plots it with a
 *                     corresponding arrow indicating a
 *                     heading "to" or a bearing "from" an
 *                     ndb or vortac.
 *
 *
 * · DOCUMENTATION
 *   FILES:            None.
 *
 *
 *   ARGUMENTS:        plane_XO            : X plot coord.
 *                     plane_YO            : Y plot coord.
 *                     plane_heading_deg   : plane heading
 *                     theta_deg           : plot angle in
 *                                           degrees
 *                     radius              : plot radius
 *                     length_ptr          : pointer to
 *                                           length of
 *                                           conv[] array
 *                     hdg_brg_conv        : pointer to
 *                                           conversion
 *                                           array
 *
 *   RETURN:           None.
 *
 *
 *   FUNCTIONS
 *   CALLED:           insert()
 *                     string_gen()
 *                     line()
 *                     arrow()
 *
 *
 *   AUTHOR:           CHUCK ROBERTSON
 *
 *
 *   DATE CREATED:     23Feb87      Version 1.0
 *
 *
```

257

```
 *   REVISIONS:          None.
 *
 *
******************************************************************/
#include<stdlib. h>
#include<string. h>
#include<math. h>
#include<stdio. h>

void heading_and_bearing(plane_X0, plane_Y0,
                         plane_heading_deg, theta_deg,
                         radius, length_ptr, hdg_brg_conv)

int plane_X0, plane_Y0;
int plane_heading_deg;
double theta_deg;
double radius;
unsigned short hdg_brg_conv[];
unsigned short *length_ptr;

{   .
    unsigned short conv[50];      /* conversion array */
    int length;                   /* insertion length */
    int i;                        /* loop counter */
    int index;                    /* array index */
    int radix;                    /* conversion radix */

    int    insert();
    int    string_gen();
    int    line();
    int    arrow();

    char final_string[20];        /* conversion strings */
    char cat_string[5];

    double size;                  /* character size */
    double theta_rad;             /* conversion angle in
                                     radians          */
    double screen_scale;          /* screen scale */
    double PI;
    double DEGREE_TO_RAD;

    int    heading_deg;           /* heading in degrees */
    int    hdg_X, hdg_Y;          /* heading plot coord */
                                  /* relative to plane  */
    int    hdg_label_X;           /* heading plot coord */
    int    hdg_label_Y;

    double hdg_label_theta;       /* final heading plot
                                     angle            */
```

258

```
double hdg_label_phi;          /* intermediate hdg plot
                                  angle               */
double hdg_label_radius;       /* hdg plot radius */
double hdg_delta_X;            /* hdg intermed x plot
                                  value               */
double hdg_delta_Y;            /* hdg intermed y plot
                                  value               */
double hdg_delta_X_sqrd;
double hdg_delta_Y_sqrd;

int    bearing_deg;            /* bearing in degrees */
int    brg_X,brg_Y;            /* bearing plot coord */
                               /* relative to plane  */
int    brg_label_X;            /* bearing plot coord */
int    brg_label_Y;

double brg_label_theta;        /* final bearing plot
                                  angle               */
double brg_label_phi;          /* intermediate brg plot
                                  angle               */
double brg_label_radius;       /* brg plot radius */
double brg_delta_X;            /* brg intermed x plot
                                  value               */
double brg_delta_Y;            /* brg intermed y plot
                                  value               */
double brg_delta_X_sqrd;
double brg_delta_Y_sqrd;

screen_scale = 9.5/12.5;
PI = 3.14159265;
DEGREE_TO_RAD = (2 * PI / 360.0);

index = 0;
size = 1.0;
radix = 10;

/* convert plot angle to radians */

theta_rad = theta_deg * DEGREE_TO_RAD;

/* calculate heading to ndb or vortac */

heading_deg = (int)(plane_heading_deg + 360
                        + (90 - theta_deg) );

if(heading_deg >= 360)
   heading_deg = heading_deg % 360;

/* calculate bearing from a ndb or vortac */
```

```
bearing_deg = (heading_deg - 180) + 360;

if(bearing_deg >= 360)
   bearing_deg = bearing_deg % 360;

/* assign delta_y constants */

hdg_delta_Y = 50.0;
brg_delta_Y = 50.0;

/* calculate delta_x values */

hdg_delta_X = 2.0 / 3.0 * radius;

brg_delta_X = 1.0 / 3.0 * radius;

/* calculate heading and bearing phi angle */

hdg_label_phi = atan2(hdg_delta_Y,
                                hdg_delta_X);

brg_label_phi = atan2(brg_delta_Y,
                                brg_delta_X);

/* calculate radii to plot heading and bearing */

hdg_delta_X_sqrd = pow(hdg_delta_X, 2.0);
hdg_delta_Y_sqrd = pow(hdg_delta_Y, 2.0);

hdg_label_radius = sqrt(hdg_delta_X_sqrd
                                + hdg_delta_Y_sqrd);

brg_delta_X_sqrd = pow(brg_delta_X, 2.0);
brg_delta_Y_sqrd = pow(brg_delta_Y, 2.0);

brg_label_radius = sqrt(brg_delta_X_sqrd
                                + brg_delta_Y_sqrd);

/* figure out which side of radius to plot heading
   and bearing                                    */

if((theta_deg <= 0.0) || (theta_deg >= 180.0))
   (hdg_label_theta = theta_rad +
                                hdg_label_phi;
    brg_label_theta = theta_rad +
                                brg_label_phi;
   )
else
   (hdg_label_theta = theta_rad -
                                hdg_label_phi;
```

260

```
    brg_label_theta = theta_rad -
                                brg_label_phi;
    }

/* calculate heading plot coordinates */

hdg_label_X =(int)(hdg_label_radius *
                            cos(hdg_label_theta) *
                            screen_scale);

hdg_label_Y =(int)(hdg_label_radius *
                            sin(hdg_label_theta));

hdg_X = hdg_label_X + plane_X0;
hdg_Y = hdg_label_Y + plane_Y0;

/* plot heading */

itoa(heading_deg,final_string,radix);
string_gen(final_string,hdg_X,hdg_Y,size,&length,conv);

for(i=0;i<length;i++)
    hdg_brg_conv[index + i] = conv[i];

index = index + length;

arrow(plane_X0,plane_Y0,theta_deg,hdg_delta_X,"to",
      &length,conv);

for(i=0;i<length;i++)
    hdg_brg_conv[index + i] = conv[i];

index = index + length;

/* calculate bearing plot coordinates */

brg_label_X =(int)(brg_label_radius *
                            cos(brg_label_theta) *
                            screen_scale);

brg_label_Y =(int)(brg_label_radius *
                            sin(brg_label_theta));

brg_X = brg_label_X + plane_X0;
brg_Y = brg_label_Y + plane_Y0;

/* plot bearing */

itoa(bearing_deg,final_string,radix);
string_gen(final_string,brg_X,brg_Y,size,&length,conv);
```

261

```
for(i=0;i<length;i++)
    hdg_brg_conv[index + i] = conv[i];

index = index + length;

arrow(plane_X0, plane_Y0, theta_deg, brg_delta_X, "from",
      &length, conv);

for(i=0;i<length;i++)
    hdg_brg_conv[index + i] = conv[i];

index = index + length;

*length_ptr = index;

}   /* end of heading_and_bearing */
```

```
/*****************************************************************
 *
 *   SOURCE FILE:      ndb_angl.c
 *
 *
 *   FUNCTION:         get_ndb_plot_angle()
 *
 *
 *   DESCRIPTION:      This routine gets the raw adf from the
 *                     data package and uses a lookup table
 *                     along with interpolation to convert
 *                     the raw adf into the ndb plot angle.
 *
 *
 *   DOCUMENTATION
 *   FILES:            None.
 *
 *
 *   ARGUMENTS:        ndb_theta_deg_ptr : pointer to the
 *                                         plot angle
 *                                         variable in
 *                                         calling routine
 *
 *
 *   RETURN:           None.
 *
 *
 *   FUNCTIONS
 *   CALLED:           None.
 *
 *
 *   AUTHOR:           CHUCK ROBERTSON
 *
 *
 *   DATE CREATED:     23Mar87      Version 1.0
 *
 *
 *   REVISIONS:        None.
 *
 *
 *****************************************************************/
#include "data_str.h"
#include<stdlib.h>

typedef struct { double a_to_d;
                 double value;  } CONVERSION;

/* lookup table for raw adf value conversion */

CONVERSION adf[18] = {
```

```
                                {0.0, 184.0},
                                {3.0, 180.0},
                                {19.0, 170.0},
                                {25.0, 150.0},
                                {47.0, 120.0},
                                {68.0, 90.0},
                                {89.0, 60.0},
                                {110.0, 30.0},
                                {131.0, 0.0},
                                {154.0, 330.0},
                                {176.0, 300.0},
                                {199.0, 270.0},
                                {221.0, 240.0},
                                {243.0, 210.0},
                                {250.0, 200.0},
                                {254.0, 190.0},
                                {255.0, 185.0}
                    };

void get_ndb_plot_angle(ndb_theta_deg_ptr)

double *ndb_theta_deg_ptr;

{
    int i;                  /* loop counter */
    double raw_adf;

    /* GET RAW ADF READING FROM A/D AND FIND PLOT ANGLE OF
       NDB IN DEGREES USING LOOKUP TABLE                    */

    raw_adf = (double)(data_pkg.ADF);

    if( (raw_adf > 131.0) && (raw_adf <= 154.0))
        adf[8].value = 360.0;
    else
        adf[8].value = 0.0;

    for(i = 0; i < 17; i++)
        if( (raw_adf >= adf[i].a_to_d) &&
            (raw_adf <= adf[i + 1].a_to_d) )
            {
                /* INTERPOLATE */
                *ndb_theta_deg_ptr

                    = (double)( (raw_adf - adf[i].a_to_d) /
                                (adf[i+1].a_to_d -
                                 adf[i].a_to_d) *
                                (adf[i+1].value -
                                 adf[i].value) +
                                 adf[i].value);
```

264

```
                break;
            }

}   /* end of get_ndb_plot_angle */
```

```
/****************************************************************
 *
 *   SOURCE FILE:      runway.c
 *
 *
 *   FUNCTION:         runway()
 *
 *
 *   DESCRIPTION:      This function calculates the HP code
 *                     to generate the runway for a given
 *                     location, angle, and scale.  It is
 *                     drawn using vanishing point
 *                     techniques.
 *
 *
 *   DOCUMENTATION
 *   FILES:            None.
 *
 *
 *   ARGUMENTS:        rho_deg               : centerline angle
 *                     x_center, y_center : plot coordinates
 *                     scale                 : runway scale
 *                     length_ptr            : pointer to length
 *                                             of conv[] array
 *                     conv                  : pointer to
 *                                             conversion array
 *
 *   RETURN:           None.
 *
 *
 *   FUNCTIONS
 *   CALLED:           None.
 *
 *
 *   AUTHOR:           CHUCK ROBERTSON
 *
 *
 *   DATE CREATED:     18Feb87      Version 1.0
 *
 *
 *   REVISIONS:        None.
 *
 *
 ****************************************************************/
#include<stdlib.h>
#include<math.h>

void runway(x_center, y_center, rho_deg, scale, length_ptr,
            runway_conv)
```

```
int x_center, y_center;
unsigned short runway_conv[];
unsigned short *length_ptr;
double rho_deg;
double scale;

{
    int     index;                      /* conversion array
                                           index           */
    int     delta_x, delta_y;           /* delta values for
                                           plot            */

    int     base_middle_X;              /* coord of middle of */
    int     base_middle_Y;              /* base of runway     */

    int     end_runway_radius;          /* radius to runway
                                           end             */

    int     end_middle_X;               /* coord of middle of */
    int     end_middle_Y;               /* end of runway      */

    int     base_left_X;                /* corner coordinates */
    int     base_left_Y;                /* of runway          */
    int     base_right_X;
    int     base_right_Y;
    int     end_left_X;
    int     end_left_Y;
    int     end_right_X;
    int     end_right_Y;

    int     base_centerline_radius;     /* radii to base and */
    int     end_centerline_radius;      /* end of runway     */
                                        /* centerline        */

    int     base_centerline_X;          /* centerline */
    int     base_centerline_Y;          /* coordinates */
    int     end_centerline_X;
    int     end_centerline_Y;
    int     touchdown_line_left_X;      /* touchdown line */
    int     touchdown_line_left_Y;      /* coordinates    */
    int     touchdown_line_right_X;
    int     touchdown_line_right_Y;

    double rho_rad;                     /* crab angle in radians  */
    double phi_deg;                     /* plot angle in degrees  */
    double phi_rad;                     /* plot angle in radians  */
    double screen_scale;
    double PI;
    double DEGREE_TO_RAD;
```

267

```
index = 0;
end_runway_radius = (int)(200 * scale);
base_centerline_radius = (int)(30 * scale);
end_centerline_radius = (int)(190 * scale);
screen_scale = 9.5/12.5;
PI = 3.14159265;
DEGREE_TO_RAD = (2 * PI / 360.0);

/* CONVERT RHO TO RADIANS */

rho_rad = rho_deg * DEGREE_TO_RAD;

/* CALCULATE MIDDLE OF BASE */

delta_x = (int)((-50 * scale) * tan(rho_rad)
               * screen_scale);
delta_y = (int)(-50 * scale);

base_middle_X = x_center + delta_x;
base_middle_Y = y_center + delta_y;

/* CALCULATE PHI */

phi_deg = 90.0 - rho_deg;

phi_rad = phi_deg * DEGREE_TO_RAD;

/* CALCULATE MIDDLE END OF RUNWAY */

delta_x = (int)(end_runway_radius * cos(phi_rad)
               * screen_scale);
delta_y = (int)(end_runway_radius * sin(phi_rad));

end_middle_X = base_middle_X + delta_x;
end_middle_Y = base_middle_Y + delta_y;

/* CALCULATE FOUR CORNERS OF RUNWAY */

base_left_X = base_middle_X -
             (int)(50 * scale * screen_scale);
base_left_Y = base_middle_Y;

base_right_X = base_middle_X +
              (int)(50 * scale * screen_scale);
base_right_Y = base_middle_Y;

end_left_X = end_middle_X -
            (int)(30 * scale * screen_scale);
end_left_Y = end_middle_Y;
```

```
end_right_X = end_middle_X +
              (int)(30 * scale * screen_scale);
end_right_Y = end_middle_Y;

/* CALCULATE RUNWAY CENTERLINE POINTS */

delta_x = (int)(base_centerline_radius *
                cos(phi_rad) * screen_scale);
delta_y = (int)(base_centerline_radius * sin(phi_rad));

base_centerline_X = base_middle_X + delta_x;
base_centerline_Y = base_middle_Y + delta_y;

delta_x = (int)(end_centerline_radius *
                cos(phi_rad) * screen_scale);
delta_y = (int)(end_centerline_radius * sin(phi_rad));

end_centerline_X = base_middle_X + delta_x;
end_centerline_Y = base_middle_Y + delta_y;

/* CALCULATE TOUCHDOWN LINE POINTS */

touchdown_line_left_X = x_center -
                        (int)(20 * scale
                              * screen_scale);
touchdown_line_left_Y = y_center;

touchdown_line_right_X = x_center +
                         (int)(20 * scale
                               * screen_scale);
touchdown_line_right_Y = y_center;

/* CONVERT TO HP CODE */

/* plot runway outline */
runway_conv[index++] = 0x0000 | base_left_X;
runway_conv[index++] = 0x1000 | base_left_Y;

runway_conv[index++] = 0x0000 | end_left_X;
runway_conv[index++] = 0x1800 | end_left_Y;

runway_conv[index++] = 0x0000 | end_right_X;
runway_conv[index++] = 0x1800 | end_right_Y;

runway_conv[index++] = 0x0000 | base_right_X;
runway_conv[index++] = 0x1800 | base_right_Y;

runway_conv[index++] = 0x0000 | base_left_X;
runway_conv[index++] = 0x1800 | base_left_Y;
```

```
        /* plot centerline */
        runway_conv[index++] = 0x0000 | base_centerline_X;
        runway_conv[index++] = 0x1000 | base_centerline_Y;

        runway_conv[index++] = 0x0000 | end_centerline_X;
        runway_conv[index++] = 0x1800 | end_centerline_Y;

        /* plot touchdown line */
        runway_conv[index++] = 0x0000 | touchdown_line_left_X;
        runway_conv[index++] = 0x1000 | touchdown_line_left_Y;

        runway_conv[index++] = 0x0000 | touchdown_line_right_X;
        runway_conv[index++] = 0x1800 | touchdown_line_right_Y;

        *length_ptr = index;

} /* end of runway */
```

```
/**************************************************************
 *
 *    SOURCE FILE:      ils_cmps.c
 *
 *
 *    FUNCTION:         ils_compass()
 *
 *
 *    DESCRIPTION:      This routine generates the HP code to
 *                      put up the hash marks and appropriate
 *                      markings for a compass showing 20
 *                      degrees at a given radius for a given
 *                      heading.
 *
 *
 *    DOCUMENTATION
 *    FILES:            None.
 *
 *
 *    ARGUMENTS:        x_center          : x plot coordinate
 *                                          for compass center
 *                      y_center          : y plot coordinate
 *                                          for compass center
 *                      heading_deg       : heading in degrees
 *                      radius            : radius of compass
 *                      length_ptr        : pointer to length of
 *                                          conv[] array
 *                      compass_conv      : pointer to
 *                                          conversion array
 *
 *    RETURN:           None.
 *
 *
 *    FUNCTIONS
 *    CALLED:           None.
 *
 *
 *    AUTHOR:           CHUCK ROBERTSON
 *
 *
 *    DATE CREATED:     30Mar87      Version 1.0
 *
 *
 *    REVISIONS:        None.
 *
 *
 **************************************************************/
#include<math.h>
#include<string.h>
#include<stdlib.h>
```

```
void ils_compass(x_center, y_center, heading_deg, radius,
                 length_ptr, compass_conv)

unsigned short compass_conv[];
unsigned short *length_ptr;
int x_center, y_center;
int radius;
int heading_deg;

(    int     X, Y;                      /* plot coordinates */
     int     i;                         /* loop counter */
     int     length;                    /* insertion length */
     int     heading_upper_bound;       /* heading bounds */
     int     heading_lower_bound;
     int     tick_value;                /* value of tick mark */
     int     num_iterations;            /* num of loop
                                           iterations  */
     int     count;                     /* loop counter */
     int     index;                     /* conversion array
                                           index         */
     int     disp;                      /* displacement for
                                           display       */
     int     str_length;                /* string insertion
                                           length        */
     int     tick_value_X0;             /* tick value plot
                                           coord         */
     int     tick_value_Y0;

     double PI;
     double delta_theta_deg;            /* angle increment
                                           degrees       */
     double delta_theta_rad;            /* angle increment
                                           radians       */
     double theta_rad, theta_deg;       /* calculation angle */
     double screen_scale;
     double DEGREE_TO_RAD;
     double start_angle_rad;            /* first calculation
                                           angle         */
     double start_angle_deg;
     double size;                       /* character size */

     char    hdg_string[3];

     unsigned short string_conv[40];

     void string_gen();

     PI = 3.14159265;
     delta_theta_deg = 1.0;
```

```
DEGREE_TO_RAD = (2 * PI / 360.0);
screen_scale = 9.5/12.5;
size = 1.0;
length = 0;

/* calculate upper and lower bounds of heading */

heading_upper_bound = (heading_deg + 10) % 360;
heading_lower_bound = (heading_deg + 350) % 360;

start_angle_deg = 100.0;

/* convert angles to radians */

delta_theta_rad = delta_theta_deg * DEGREE_TO_RAD;
start_angle_rad = start_angle_deg * DEGREE_TO_RAD;

/* INITIALIZE CALCULATION ANGLE, TICK_VALUE, INDEX,
   AND # OF ITERATIONS */

theta_rad = start_angle_rad;
tick_value = heading_lower_bound;
index = 0;
num_iterations = 21;

/* CALCULATE COORDINATES AND CHARACTERS FOR HP CODE */

for(count = 0; count < num_iterations; count++)
   {
      X = (int)(radius * cos(theta_rad) * screen_scale);
      Y = (int)(radius * sin(theta_rad));

      compass_conv[index++] = 0x0000 | (X + x_center);
      compass_conv[index++] = 0x1000 | (Y + y_center);

      if((tick_value % 5) == 0)
         disp = 100;
      else
         disp = 50;

      X = (int)((radius - disp) * cos(theta_rad)
                * screen_scale);
      Y = (int)((radius - disp) * sin(theta_rad));

      compass_conv[index++] = 0x0000 | (X + x_center);
      compass_conv[index++] = 0x1800 | (Y + y_center);

      theta_rad = theta_rad - delta_theta_rad;

      if((tick_value % 90) == 0)
```

273

```
    (
        switch (tick_value)
          (
              case 0:
                strcpy(hdg_string, "N");
                break;
              case 90:
                strcpy(hdg_string, "E");
                break;
              case 180:
                strcpy(hdg_string, "S");
                break;
              case 270:
                strcpy(hdg_string, "W");
                break;
              default:
                break;
          )
        str_length = 1;
    )
else   if((tick_value % 10) == 0)
          (
              itoa(tick_value, hdg_string, 10);
              str_length = strlen(hdg_string);
          )

/* CALCULATE COORDINATES FOR TICK_VALUE */

switch (str_length)
   (
       case 1:
          disp = 18;
          break;
       case 2:
          disp = 36;
          break;
       case 3:
          disp = 54;
          break;
   )

tick_value_X0 = X + x_center - disp;
tick_value_Y0 = Y + y_center - 50;

if((tick_value % 10) == 0)
  (
      string_gen(hdg_string, tick_value_X0,
                 tick_value_Y0, size, &length,
                 string_conv);
```

274

```
            for(i=0;i<length;i++)
                compass_conv[index + i] = string_conv[i];

            index = index + length;
        }

        tick_value = (tick_value + 1) % 360;

    }

*length_ptr = index;

} /* end ils_compass */
```

APPENDIX D (CONT.)

HOST PROGRAM INCLUDE FILES

```
/**************************************************************
 *
 *   SOURCE FILE:    data_str.h
 *
 *
 *   FUNCTION:       None.
 *
 *
 *   DESCRIPTION:    This is a file to be included in any
 *                   function that needs to access a member
 *                   of the data package. All the pieces of
 *                   the data package are stored in the
 *                   structure data_pkg.  Individual members
 *                   are accessed by using the following
 *                   name: data_pkg.ALTITUDE   etc.
 *
 *
 *   DOCUMENTATION
 *   FILES:          None.
 *
 *
 *   ARGUMENTS:      None.
 *
 *
 *   RETURN:         None.
 *
 *
 *   FUNCTIONS
 *   CALLED:         None.
 *
 *
 *   AUTHOR:         Dave Gruenbacher & Chuck Robertson
 *
 *
 *   DATE CREATED:   22Jan87      Version 1.0
 *
 *
 *   REVISIONS:
 *
 *                   13Apr87      Version 1.1
 *                   Added conditional statements.
 *                   Dave Gruenabcher
 *
 **************************************************************/

typedef struct {
unsigned char   BANK                                    ;
unsigned char   PITCH                                   ;
unsigned char   VERT_SPEED                              ;
unsigned char   DELTA_X                                 ;
```

```
unsigned char   DELTA_Y                               ;
unsigned char   MANIFOLD_PRESSURE ;
unsigned char   COURSE_DEVIATION            ;
unsigned char   GLIDESLOPE                       ;
unsigned char   ALTITUDE                         ;
unsigned char   AIRSPEED                         ;
unsigned char   COMPASS                          ;
unsigned char   ADF                                    ;
unsigned char   DME                                    ;
unsigned char   POWER                            ;
unsigned char   RPM                                    ;
unsigned char   SPARE                            ;
unsigned char   BINARY_INPUTS            ;
unsigned char   LAST_KEY                         ;
unsigned char   MONTH                            ;
unsigned char   DAY                                    ;
unsigned char   DATE                             ;
unsigned char   HOURS                   ;
unsigned char   MINUTES                          ;
unsigned char   SECONDS                          ;
                                    } DATA_PKG;


typedef struct {
                int     timer_min;
                int     timer_sec;
                int     time_out_min;
                int     time_out_sec;
                int     math_operation_flag;
                int     timer_operation_flag;
                int     timer_status_flag;
                double  adf_freq;
                double  com1_freq;
                double  com2_freq;
                double  vor1_freq;
                double  vor2_freq;
                int     assigned_altitude;
                int     mda_dh;
                int     estimated_wind;
                            } CLOCK_PKG;

typedef struct {
                int     airspeed_alarm_flag;
                int     assigned_altitude_alarm_flag;
                int     mda_dh_alarm_flag;
                int     assigned_altitude_enable_flag;
                int     mda_dh_enable_flag;
                int     time_out_alarm_flag;
                int     alarm_status_flag;
                                    } ALARM_PKG;
```

```
#ifdef ehsi_main
DATA_PKG data_pkg = {0};
unsigned short SCREEN[1000] = {OxFFFF};
#else
extern DATA_PKG data_pkg;
extern unsigned short SCREEN[];
extern int GET_DATA_PACKAGE();
extern int SEND_SCREEN();
extern int RETRIEVE_SCREEN();
extern int TOGGLE_ALARM_SWITCH();
#endif

#define    NULL_TIMER     0
#define    START_TIMER    1
#define    RESET_TIMER    2
#define    SET_TIMER      3
#define    TIMER_OFF      0
#define    TIMER_ON       1

#define    ALARM_OFF      0
#define    ALARM_ON       1

#define    DISABLED       0
#define    ENABLED        1
```

```
/******************************************************************
 *
 *   SOURCE FILE:       datpg_xy.c
 *
 *
 *   FUNCTION:          None.
 *
 *
 *   DESCRIPTION:       This file is to be included with the
 *                      any file dealing with the DATA PAGE
 *                      that needs to display something on the
 *                      vector graphics display.  It contains
 *                      coordinates for plotting the various
 *                      items on the DATA PAGE.
 *
 *
 *   DOCUMENTATION
 *   FILES:             None.
 *
 *
 *   ARGUMENTS:         None.
 *
 *
 *   RETURN:            None.
 *
 *
 *   FUNCTIONS
 *   CALLED:            None.
 *
 *
 *   AUTHOR:            CHUCK ROBERTSON
 *
 *
 *   DATE CREATED:      01Feb87      Version 1.0
 *
 *
 *   REVISIONS:         None.
 *
 *
 ******************************************************************/
#define col1 10
#define col2 1100

#define heading_X0 col1
#define heading_Y0 1980
#define heading_X1 col1 + (10 * 54)
#define heading_Y1 1980

#define airspeed_CAS_X0 col1
#define airspeed_CAS_Y0 1890
```

```
#define airspeed_CAS_X1 col1 + (10 * 54)
#define airspeed_CAS_Y1 1890

#define airspeed_TAS_X0 col1
#define airspeed_TAS_Y0 1800
#define airspeed_TAS_X1
#define airspeed_TAS_Y1

#define gndspeed_X0 col1
#define gndspeed_Y0 1710
#define gndspeed_X1
#define gndspeed_Y1

#define assigned_X0 col1
#define assigned_Y0 1530
#define assigned_X1 col1 + (10 * 54)
#define assigned_Y1 1530

#define altitude_X0 col1
#define altitude_Y0 1440
#define altitude_X1 col1 + (10 * 54)
#define altitude_Y1 1440

#define mda_dh_X0 col1
#define mda_dh_Y0 1350
#define mda_dh_X1 col1 + (8 * 54)
#define mda_dh_Y1 1350

#define timer_X0 col1
#define timer_Y0 1170
#define timer_X1 col1 + (7 * 54)
#define timer_Y1 1170

#define time_out_X0 col1
#define time_out_Y0 1080
#define time_out_X1 col1 + (9 * 54)
#define time_out_Y1 1080

#define com1_X0 col1
#define com1_Y0 900
#define com1_X1 col1 + (6 * 54)
#define com1_Y1 900

#define com2_X0 col1
#define com2_Y0 810
#define com2_X1 col1 + (6 * 54)
#define com2_Y1 810

#define nav1_X0 col1
#define nav1_Y0 720
```

281

```c
#define nav1_X1 col1 + (6 * 54)
#define nav1_Y1 720

#define nav2_X0 col1
#define nav2_Y0 630
#define nav2_X1 col1 + (6 * 54)
#define nav2_Y1 630

#define rnav_wp1_X0 col1
#define rnav_wp1_Y0 450
#define rnav_wp1_X1
#define rnav_wp1_Y1

#define rnav_wp2_X0 col1
#define rnav_wp2_Y0 360
#define rnav_wp2_X1
#define rnav_wp2_Y1

#define adf_X0 col2
#define adf_Y0 450
#define adf_X1 col2 + (5 * 54)
#define adf_Y1 450

#define temp_X0 col1
#define temp_Y0 180
#define temp_X1
#define temp_Y1

#define time_X0 col2
#define time_Y0 900

#define time_edt_X0 time_X0
#define time_edt_Y0 time_Y0 - 90
#define time_edt_X1 time_X0 + (10 * 52)
#define time_edt_Y1 time_Y0 - 90

#define time_zulu_X0 time_X0
#define time_zulu_Y0 time_Y0 - 180
#define time_zulu_X1 time_X0 + (10 * 54)
#define time_zulu_Y1 time_Y0 - 180

#define time_since_lo_X0 time_X0
#define time_since_lo_Y0 time_Y0 - 270
#define time_since_lo_X1 time_X0 + (10 * 54)
#define time_since_lo_Y1 time_Y0 - 270

#define barometer_X0 col2
#define barometer_Y0 180
#define barometer_X1
#define barometer_Y1
```

```c
#define since_lo_X0 col1
#define since_lo_Y0 630
#define since_lo_X1
#define since_lo_Y1

/* coordinates for "CLIMB" */
#define climb_arc_X0 col2 + 100
#define climb_arc_Y0 1440 + 27

#define vert_speed_X0 climb_arc_X0
#define vert_speed_Y0 climb_arc_Y0 + 270 - 27

#define C_X0 climb_arc_X0 - 100
#define C_Y0 climb_arc_Y0 + 180 - 27

#define L_X0 climb_arc_X0 - 100
#define L_Y0 climb_arc_Y0 + 90  - 27

#define I_X0 climb_arc_X0 - 100
#define I_Y0 climb_arc_Y0 - 27

#define M_X0 climb_arc_X0 - 100
#define M_Y0 climb_arc_Y0 - 90 - 27

#define B_X0 climb_arc_X0 - 100
#define B_Y0 climb_arc_Y0 - 180 - 27

#define climb_centerline1_X0 climb_arc_X0 - 47
#define climb_centerline1_Y0 climb_arc_Y0
#define climb_centerline1_X1 climb_arc_X0 + 47
#define climb_centerline1_Y1 climb_arc_Y0

#define climb_centerline2_X0 climb_arc_X0 - 47 + 190
#define climb_centerline2_Y0 climb_arc_Y0
#define climb_centerline2_X1 climb_arc_X0 + 47 + 190
#define climb_centerline2_Y1 climb_arc_Y0

#define climb_centerline3_X0 climb_arc_X0 - 47 + 2 * 190
#define climb_centerline3_Y0 climb_arc_Y0
#define climb_centerline3_X1 climb_arc_X0 + 47 + 2 * 190
#define climb_centerline3_Y1 climb_arc_Y0

#define climb_centerline4_X0 climb_arc_X0 - 47 + 3 * 190
#define climb_centerline4_Y0 climb_arc_Y0
#define climb_centerline4_X1 climb_arc_X0 + 47 + 3 * 190
#define climb_centerline4_Y1 climb_arc_Y0

#define climb_centerline5_X0 climb_arc_X0 - 47 + 4 * 190
#define climb_centerline5_Y0 climb_arc_Y0
#define climb_centerline5_X1 climb_arc_X0 + 47 + 4 * 190
```

```
#define climb_centerline5_Y1 climb_arc_Y0

/* end of coordinates for "CLIMB" */

#define aidspeed_bar_X0 900
#define aidspeed_bar_Y0 1800

#define binary_ip_X0 col2
#define binary_ip_Y0 300

#define command_line_X0 10
#define command_line_Y0 160
#define command_line_X1 2038
#define command_line_Y1 160

#define command_X0 10
#define command_Y0 10
```

```
/****************************************************************
 *
 *   SOURCE FILE:      navpg_xy.c
 *
 *
 *   FUNCTION:         None.
 *
 *
 *   DESCRIPTION:      This file is to be included with the
 *                     any file dealing with the NAV PAGE
 *                     that needs to display something on
 *                     the vector graphics display.   It
 *                     contains coordinates for plotting the
 *                     various items on the NAV PAGE.
 *
 *
 *   DOCUMENTATION
 *   FILES:            None.
 *
 *
 *   ARGUMENTS:        None.
 *
 *
 *   RETURN:           None.
 *
 *
 *   FUNCTIONS
 *   CALLED:           None.
 *
 *
 *   AUTHOR:           CHUCK ROBERTSON
 *
 *
 *   DATE CREATED:     05Apr87      Version 1.0
 *
 *
 *   REVISIONS:        None.
 *
 *
 ****************************************************************/
#define airspeed_X0 10
#define airspeed_Y0 1950
#define airspeed_X1 10 + (7 * 54)
#define airspeed_Y1 1950

#define gndspeed_X0 10
#define gndspeed_Y0 1850
#define gndspeed_X1
#define gndspeed_Y1
```

```
#define track_X0 1600
#define track_Y0 1950
#define track_X1
#define track_Y1

#define altitude_X0 1600
#define altitude_Y0 1850
#define altitude_X1 1600 + (4 * 54)
#define altitude_Y1 1850

#define dme_X0 10
#define dme_Y0 1750
#define dme_X1 10 + (5 * 54)
#define dme_Y1 1750

#define plane_X0 1048
#define plane_Y0 855

#define compass_arc_X0           1048
#define compass_arc_Y0           10
#define compass_arc_init_angle 45.0
#define compass_arc_end_angle  135.0
#define compass_arc_radius       1860

/* HEADING, HEADING BOX, AND POINTER */

#define heading_X0 967
#define heading_Y0 1980

#define heading_box_X0 heading_X0
#define heading_box_Y0 heading_Y0
#define heading_box_no_char 3

#define heading_ptr_line1_X0 heading_X0 - 10
#define heading_ptr_line1_Y0 heading_Y0 - 10
#define heading_ptr_line1_X1 compass_arc_X0
#define heading_ptr_line1_Y1 compass_arc_Y0 + \
                             compass_arc_radius

#define heading_ptr_line2_X0 (heading_X0 +                    \
                             (heading_box_no_char * 54) + \
                             10)
#define heading_ptr_line2_Y0 heading_Y0 - 10
#define heading_ptr_line2_X1 compass_arc_X0
#define heading_ptr_line2_Y1 compass_arc_Y0 + \
                             compass_arc_radius

#define command_line_X0 10
#define command_line_Y0 160
#define command_line_X1 2038
```

```
#define command_line_Y1 160

#define command_X0 10
#define command_Y0 10
```

```
/*****************************************************************
 *
 *   SOURCE FILE:      ilspg_xy.c
 *
 *
 *   FUNCTION:         None.
 *
 *
 *   DESCRIPTION:      This file is to be included with the
 *                     any file dealing with the ILS PAGE
 *                     that needs to display something on the
 *                     vector graphics display.  It contains
 *                     coordinates for plotting the various
 *                     items on the ILS PAGE.
 *
 *
 *   DOCUMENTATION
 *   FILES:            None.
 *
 *
 *   ARGUMENTS:        None.
 *
 *
 *   RETURN:           None.
 *
 *
 *   FUNCTIONS
 *   CALLED:           None.
 *
 *
 *   AUTHOR:           CHUCK ROBERTSON
 *
 *
 *   DATE CREATED:     05Apr87      Version 1.0
 *
 *
 *   REVISIONS:        None.
 *
 *
 *****************************************************************/
#define airspeed_X1 0
#define airspeed_Y1 0

#define altitude_X0 10
#define altitude_Y0 1950
#define altitude_X1 10 + (5 * 54)
#define altitude_Y1 1950

#define mda_dh_X0 10
#define mda_dh_Y0 1850
```

```
#define mda_dh_X1 10 + (4 * 54)
#define mda_dh_Y1 1850

#define dme_X0 1600
#define dme_Y0 1950
#define dme_X1 1600 + (5 * 54)
#define dme_Y1 1950

#define compass_arc_X0           1048
#define compass_arc_Y0           10
#define compass_arc_init_angle 80.0
#define compass_arc_end_angle  100.0
#define compass_arc_radius       1860

/* HEADING, HEADING BOX, AND POINTER */

#define heading_X0 967
#define heading_Y0 1980

#define heading_box_X0 heading_X0
#define heading_box_Y0 heading_Y0
#define heading_box_no_char 3

#define heading_ptr_line1_X0 heading_X0 - 10
#define heading_ptr_line1_Y0 heading_Y0 - 10
#define heading_ptr_line1_X1 compass_arc_X0
#define heading_ptr_line1_Y1 compass_arc_Y0 + \
                             compass_arc_radius

#define heading_ptr_line2_X0 heading_X0 + \
                             (heading_box_no_char * 54) + 10
#define heading_ptr_line2_Y0 heading_Y0 - 10
#define heading_ptr_line2_X1 compass_arc_X0
#define heading_ptr_line2_Y1 compass_arc_Y0 + \
                             compass_arc_radius

#define command_line_X0 10
#define command_line_Y0 160
#define command_line_X1 2038
#define command_line_Y1 160

#define command_X0 10
#define command_Y0 10
```

GLOSSARY

ADF - Automatic Direction Finder. Used for finding the direction of a NDB which is transmitting the set ADF frequency. Usually shown as an angle relative to the planes heading.

Bearing - The horizontal direction from a point measured clockwise from some reference point, usually magnetic north, through 360 degrees.

DACI - Data Acquisition and Communications Interface. The interface unit, which ties the EHSI system together and communicates with the host computer.

DH - Decision Height. Height at which a decision must be made, during an ILS approach, to either continue the approach or to execute a missed approach.

DME - Distance Measuring Equipment. Equipment used to measure, in nautical miles, the slant range distance of an aircraft from the DME navigation aid.

Glideslope - Descent profile determined for vertical guidance during an approach.

Groundspeed - The speed of the airplane relative to the surface of the earth.

Heading - The horizontal direction to a point measured clockwise from some reference point, usually magnetic north, through 360 degrees.

IFR - Instrument Flight Rule. Rules which apply when flying an aircraft under instrument conditions.

ILS - Instrument Landing System. System used when landing an aircraft under IFR conditions. Contains a glide-slope, localizer, and markers.

Localizer - The component of an ILS, which provides course guidance to the runway.

MDA - Minimum Descent Altitude. Lowest altitude to which descent is authorized on final approach in execution of a standard instrument approach procedure where no electronic glideslope is provided.

290

GLOSSARY

NDB - Non-directional Beacon.   A radio beacon transmitting
    non-directional signals.   Direction to or   from found
    using ADF equipment.

Track - The actual flight path of an aircraft over the
    surface of the earth.

VORTAC - VHR Omnidirectional Range Tactical Air Navigation.
    A navigational aid providing VOR azimuth, TACAN azimuth,
    and TACAN DME equipment at one location.

Waypoint - A predetermined geographical position used for
    route progress that is defined relative to a VORTAC.

GRAPHICAL PAGE DEVELOPMENT OF AN
ELECTRONIC HORIZONTAL SITUATION INDICATOR

by

CHARLES A. ROBERTSON

B.S., Kansas State University, 1985

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1987

ABSTRACT

A brief introduction to an Electronic Horizontal Situation Indicator (EHSI) is given. Development of the graphical pages of the EHSI is discussed. An overview of the development system used is given. The controlling routine of the EHSI is discussed along with graphical routines used to generate the various pages of the EHSI. Suggestions are made for improving the EHSI Development System. Future recommendations for further development of the EHSI pages are given.